UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Tesis Final de Máster

Diseño de interfaz entre una placa Beaglebone y periféricos mediante el bus GPMC

por

Eugenia Suárez Vicente

Director: Manuel Domínguez Pumar

30 ECTs de la titulación de Máster en Ingeniería Electrónica e
Ingeniería Electrónica en Telecomunicaciones en:
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

Barcelona, mayo 2016

Resumen

El propósito de este trabajo es proporcionar una primera versión de interfaz funcional entre un periférico de uno o varios tipos de memoria externa y una Beaglebone soportando una distribución de Linux diferente a la que sostenía por defecto. El proyecto forma parte del desarrollo de un sistema low cost para ser empleado durante las prácticas de la asignatura *DSX* y formar parte de su contenido. El objetivo es obtener las señales de control del protocolo de acceso a dicha memoria, como también lo es conocer las peculiaridades y diferencias más significativas entre estos tipos de memoria (NOR y NAND).

BeagleBone es una de las plataformas de sistemas embebidos del mercado conocidas como minicromputadoras. Se caracterizan por su bajo coste, portabilidad, potencia de procesamiento, conjuntos de I/O e, incluso, ampliabilidad. Debido a que Angstrom, distribución de OS actual, deja de tener soporte oficial, el diseño deberá ser adaptado a la de Ubuntu.

El desarrollo se compone de un diseño hardware, para el elemento periférico y otra de programación del driver GPMC, en el lado de la CPU. Se puede resumir en 4 fases: preparación del entorno de desarrollo (incluída migración de OS), diseño de circuito impreso PCB (integración de periférico), programación del driver GPMC e integración de todos los elementos (comprobación de la interfaz). Tras los resultados, se comentarán las limitaciones y las perspectivas que suscita.

Dedico este trabajo a mis amados padres por la responsabilidad y papel que tienen tanto su apoyo, como perseverancia, sacrificio y esfuerzos en todas las facetas de mi vida y en que este trabajo haya visto la luz.

Agradecimientos

A Manuel por darme la oportunidad de participar en este proyecto, mejorar mi bagaje y ampliar mi experiencia aprendiendo todo lo que he aprendido realizando este trabajo.

A Juan por sus consejos y disponibilidad cuando más dudas tuve.

A Rubén, por sus ánimos, paciencia, dedicación y diligencia a la hora de ofrecer desinteresadamente su soporte.

Por la calidad humana y comprensión que han añadido a su labor quiero hacer mención especial a:

al personal de secretaría por su atención y asesoramiento prestado,

al equipo encargado de los laboratorios del sótano, especialmente a Manel y Vicente, que han tenido una gran peso en disponer del entorno de trabajo más apropiado y afín a mis intereses académicos,

Sandra Bermejo por su asesoramiento, consejos y ánimos.

A compañeros como Carlos, siempre dispuesto a debatir y comentar cualquier dilema electrónico.

Todos han influido y contribuido en cierta medida a facilitar el trabajo.

A toda mi familia de sangre y a mi otra familia, la de los amigos por su apoyo incondicional, paciencia impagable y esperanza en que este momento llegara. A personas como Isabel, Gisela, Marta y Almudena por su hospitalidad durante este trabajo cuando más me hizo falta. A todos aquellos seres queridos que han aguardado pacientemente mi asuencia durante toda mi formación y no les he perdido.

Gracias

Índice

Resumen 2

Agradecimientos 3

Índice 4

Lista de figuras 6

List de tablas 8

1. **Introducción** 9
    1. **Contexto** 9
    2. **Objetivos y especificaciones** 10
    3. **Trabajo realizado anteriormente** 11
    4. **Sistemas embebidos** 12
    5. **Memorias flash; tecnología NOR vs NAND en sistemas embebidos** 13
    6. **Bus GPMC** 15
2. **Estado del arte y motivaciones para la elección de la tecnología empleada** 18
    1. **Minicomputers** 18
        1. **Intel Edison** 18
        2. **Raspberry** 18
        3. **ODROID-C2** 19
        4. **Arduino** 19
        5. **Beaglebone White** 20
        6. **Beaglebone Black** 21
        7. **Modelo empleado** 21
            1. **CAPE-BONE-TT01V** 21
            2. **CAPE GPMC interface** 22
    2. **Dispositivos Flash Nand** 22
        1. **Soluciones para conectar una NAND a un procesador RISC** 24
        2. **Principales fabricantes y productos analizados** 25
            1. **Spansion/Cypress** 25
            2. **Toshiba** 26
            3. **Hynix** 27
            4. **Samsung** 27
            5. **Micron** 27
        3. **Elección del modelo** 28

1. Arquitectura y funcionamiento....29
2. Bus en alta impedancia....31
3. Metodología y desarrollo....32
1. Preparación del entorno de desarrollo....32
2. Diseño de circuito impreso PCB....35
1. Eleccion de software CAD de diseno PCB....35
2. Familiarización con el *soft cad*....35
3. Restricciones y criterios de fabricacion.... 38
1. Deducción de valor de Rpull up....39
2. Filtro de acoplo....40
4. Lista de componentes para adaptación de dispositivo....40
5. Diseno final PCB; cara top, cara bottom....40
6. Generacion de ficheros GERBER para la fabricacion....42
7. Comprobacion continuidad de las pistas trazadas....43
3. Programación del driver GPMC....44
1. Memory Slide: Chip de memoria NAND MT29F4G16ABADAWP....44
1. Consideraciones 44
2. Sincronismo de control de la memoria según su fabricante....45
2. Host Driver Side....53
4. Integración de los elementos de la interfaz....68
4. Resultados 69
1. Comprobacion del modulo GPMC....69
1. Inicializacion del driver y dispositivo periferico....70
2. Operacion de escritura....71
3. Operacion de lectura....72
2. Integración....72
1. Lectura de ID....73
5. Conclusiones....74
6. Bibliografia....77
7. Anexos....79
8. Glosario....97

Lista de figuras

Fig. 1: Beaglebone y componente periférico a comunicar....10

Fig 2: Comparación de arquitecturas entre ambas tecnologías....14

Fig 3: Cuadro resumen de NOR vs NAND....15

Fig 4 : Relación de capacidades y tiempos en función de la tecnología....15

Fig 5: a) Integración del driver GPMC en el uP, b) interfaz GPMC – memoria sync, c) GPMC – NAND....16

Fig 6: Señales de control del bus GPMC....17

Fig 7: Ilustración de la CAPE-BONE-TT01V1....22

Fig 8: Evolución de la demanda de NAND Flash del mercado....23

Fig 9: Organización de una matriz NAND de 2 GB....23

Fig 10: Ejemplo de una operación de lectura de página; multiplexación en tiempo del bus I/O....24

Fig 11: Comparativo entre interfaz Row y eMMC....24

Fig 12: Cuota de mercado mundial por fabricantes de memoria NAND Flash (2010 – 2015)....25

Fig 13: Petición de operación NAND....29

Fig 14: Arquitectura del dispositivo empleado....30

Fig 15: Transferencia de datos en lectura/escritura de una página....30

Fig. 16: Buffer triestado....31

Fig.17: Interfaz Raw....32

Fig 18: Archivo configuración ethernet en host portátil....33

Fig 19: Trazado automático de signals....35

Fig 20: Cabeceras y contacto....36

fig 21: Puenteo WP y VCC en la Cape-bone-TT01v....38

Fig 22: Correspondencia de pines CN1 – NAND....38

Fig 23: Circuito en drenador abierto de la señal RB....39

Fig 24:tiempo de caída y de subida de RB en tC y tC en función de Rpull up....40

Fig 25: caras top y bottom del diseño final....41

Fig 26: generacion de archivo de la capa bottom....42

Fig 27: a) Línea de comando para generar taladros....43

b) directorio de archivos de fabricación generados

Fig 28: a) ciclo de command latch, b) ciclo de address latch....45

Fig 29: a) ciclos de escritura, b) ciclos de lectura....46

Fig 30: Lanzamiento comando RESET....48

Fig 31: Operación comando read status....48

Fig 32: Sincronismo de lectura de ID....50

Fig 33: Sincronismo de activación de una lectura de página....50

Fig 34: Sincronismo de borrado de bloque....51

Fig 35: Sincronismo de una operación de programación de página....52

Fig 36: Fases de programación para implementar driver flash GPMC....54

Fig 37: Declaración de las dos variables struct....55

Fig. 38: Mapeo del grupo de registros CM_PER (del mapa de memoria de la interfaz de periféricos L4_WKUP)....56

Fig 39: Offset de registros del grupo CM_PER....57

Fig. 40: Mapeo y baseaddress del Control Module....57

Fig 41: Mapeo de los registros GPMC en L3....57

Fig 42: L3 GPMC memory map: espacio reservado para acceso a memoria externa....57

Fig 43: Formas de onda GPMC para el ciclo de comando....58

Fig. 44: Formas de onda GPMC para un ciclo de dirección....59

Fig 45: Ciclo escritura datos....59

Fig. 46: Ciclo lectura datos....60

Fig. 47: bits 6:0 del registro conf_<module>_<pin> 65

Fig. 48: a) cara bottom (inferor), b) cara top (superior)....68

Fig. 49: Relación CN2 con señales analizador digital....69

Fig. 50: Comportamiento del driver cuando no se ha cargado en el sistema....70

Fig. 51: Carga del driver....70

Fig. 52: Zoom de la imagen anterior: secuencia de inicialización del driver....70

Fig. 53:secuencia completa de una escritura de página....70

Fig. 54:secuencia completa de una lectura de página....71

Fig. 55: Secuencia de accesos en la interfaz para leer ID de la NAND....72

Lista de tablas:

Tabla 1: resumen características Intel Edison....18

Tabla 2: resumen características Rasp Pi B+....18

Tabla 3.resumen características Rasp Pi 3....19

Tabla 4: resumen características ODROID-C2....19

Tabla 5: resumen características ArduinoUno....20

Tabla 6: resumen características Arduino Yun....20

Tabla 7: resumen características Arduino Due....20

Tabla 8: resumen características Beaglebone White....21

Tabla 9:Partición de dirección completa en direcciones parciales en una memoria de 2GB....24

Tabla 10:Familia NAND cypress....26

Tabla 11: Tipos de NAND Toshiba....27

Tabla 12: Gama de NAND Micron....28

Tabla 13: Ejemplos de modelos SLC....28

Tabla 14: OrCad Vs. Eagle light Ed....35

Tabla 15: Seguimiento de señales entre BB (P8, P9) y la Cape-bone-TT01v1....37

Tabla 16: lista de componentes definitiva....40

Tabla 17: Tiempos de setup y de hold para comandos....46

Tabla 18: Tiempos para la escritura de dirección....47

Tabla 19: Tiempos del comando de reset....48

Tabla 20: Parámetros temporales para la lectura de estado....49

Tabla 21: Significado de los campos del registro de estado de la memoria....49

Tabla 22: Tiempos de una lecura de página ….....50

Tabla 23: Tiempos de una escritura de página....52

Tabla 24: Resumen de las etapas de la operación de escritura y de lectura que el host debe implentar 53

Tabla 25: Lista de los registros de configuración del driver GPMC....54

Tabla 26: Pasos de solución avanzada Vs. solución básica 56

Tabla 27: Parámetros de sincronismo en los 3 tipos de ciclos de escritura....58

Tabla 28: Parámetros de sincronismo de la lectura....60

Tabla 29: Relación de operaciones internas cuando se ejecuta acceso a un registro de datos....62

Tabla 30: Lista de comandos implementados en la versión del driver actual....67

Tabla 31: Valor de la dirección que se asigna desde el código....67

Tabla 32: Direccionamiento NAND: Representación numérica de los valores del bus durante los ciclos dedireccionamiento de la imagen anterior (Horizontal Bits del bus de datos/ Vertical número de ciclo)....71

1.- Introducción

La familia de Beagleboards forma parte de una oferta de placas consideradas en buena parte como placas de entrenamiento: arduino, Raspberry, Orange Pi, Intel Edison, etc. Como tales, constituyen una herramienta de desarrollo muy eficaz entre diseñadores y de aprendizaje fundamental en el ámbito docente.

Beaglebone White es una tarjeta de tamaño reducido basada en un procesador ARM de la familia Sitara que ha sido suministrada con una preinstalación del SO Angtrom. Dispone de un surtido módulos para ampliar su conectividad llamados CAPEs. Esta placa forma parte del material de prácticas de una asignatura *Digital Systems Using Embedded Linux (DSX)* impartida en la universidad UPC a cargo del departamento de Ingeniería Electrónica. En las sesiones prácticas se provee al alumnado con una CAPE usando un software y aplicaciones particulares para obtener su funcionalidad, junto con la placa.

Uno de los estándares de comunicación con periféricos implementados en la placa es el bus GPMC, que gracias a la CAPE y a la programación de su driver se puede interconectar a un dispositivo de memoria externo compatible. Este es uno de los objetivos del plan de la asignatura. Hasta ahora, las comunicaciones a través de este estándar estaban orientadas al acceso de una memoria emulada en el hardware de una FPGA, debido a la flexibilidad que permite trabajar describiendo hardware. Sin embargo, la CAPE presenta un conector adaptado para las comunicaciones NOR. Aunque no es difícil readaptar el conector a un acceso NAND, la complejidad de una descripción de harware de una memoria NAND se eleva exponencialmente y, por lo tanto, la viabilidad de esta vía dentro de los márgenes de tiempo restante resulta incierta.

1.1.- Contexto:

La motivación de este trabajo es la de ampliar la gama de periféricos accesibles desde la Beagleboard en el contexto de las prácticas de la asignatura. Partiendo de un driver adaptado al nuevo sistema se desarrollorá un controlador para flash NAND -que sólo puede ser asíncrona y secuencial- y una alternativa para flash NOR.

Debido a la ausencia de soporte técnico actual sobre el sistema preinstalado se determinó reemplazar SO Angstrom por el SO Ubuntu y una adaptación del software de las CAPEs al mismo, verificado en un trabajo anterior. De modo que el primer paso para iniciar el desarrollo del presente trabajo consta en seguir las indicaciones sugeridas para una correcta migración hacia dicho sistema (se validará probando aplicaciones ya adaptadas y verificadas). A partir de ahí se continuará con la instalación del entorno de trabajo. Se desarrollarán los códigos en C: adaptación del código del driver GPMC para manejar accesos flash NAND y un sencillo algoritmo que permita al usuario demandar estas operaciones al driver.

En paralelo se procederá a una valoración de las familias de memorias flash NAND del mercado y elección del fabricante y modelo. Una vez seleccionado, se diseñará un esquemático con las indicaciones y sugerencias del fabricante. El siguiente paso es realizar el diseño de la PCB mediante una herramienta CAD, que más se adapte a las necesidades y cumpla los requisitos mínimos del trabajo.

Las memorias Flash NAND y Flash NOR son dos tecnologías basadas en el principio flash (matriz de puertas lógicas negadas) pero diametralmente opuestas en varios aspectos. Hasta ahora, debido a su naturaleza, esas diferencias, han dividido el uso de las dos tecnologías en dos campos de aplicaciones. El acceso NOR es aleatorio (random) y el acceso NAND es secuencial y en streaming. Por lo que NOR solía ser la más empleada en almacenamiento de memoria de programa, mientras que NAND estaba siendo idónea para almacenamiento de dispositivos que capturan ráfagas de grandes de datos de forma compacta. Sin embargo los avances tecnológicos permiten emplear una memoria NAND en modo boot, en lugar de NOR. El proceso es algo más complejo, pero se obtienen las ventajas de una memoria NAND. No obstante la diferencia trasciende de su aplicación. El direccionamiento y los accesos de estos dos tipos de memoria son muy diferente y debe ser cuidadosamente definidos en el software de control. Disponer de las dos alternativas en el material de la asignatura es una propuesta

completa y enriquecedora. Académicamente, merece la pena conocerlas y tener contacto no sólo teórico sino práctico con sus diferencias -y similitudes-.

Finalmente se ofrecerá un interesante rango de posibilidades de maduración de la interfaz que brindaría la incorporación de este material -en un punto de partida mejorable- a la asignatura. También se expondrán las limitaciones que el contexto de este proyecto propició.

El manual técnico de referencia del procesador, un AM335xx de Texas instruments y su foro oficial disponen la información sobre cómo programar el driver GPMC en el apartado dedicado al sistema de memoria. El manual refiere a una serie de registros de configuración cuyos campos deben completarse con los valores apriopiadamente definidos, sobretodo las temporizaciones de las señales de control, en el lado CPU (host/beaglebone) de la interfaz a crear. Mientras que la otra gran parte de información requerida se consultará en el datasheet del modelo de memoria seleccionada que ayudará a deducir estos timings, el direccionamiento y las fases necesarias para completar las operaciones de acceso.

1.2- Objetivos y especificaciones:

El objetivo es obtener una interfaz funcional capaz de dar una solución a la comunicación de la Beagbleone y su CAPE (a través del conector del bus GPMC existente en la CAPE) con un modelo de flash NAND específico.

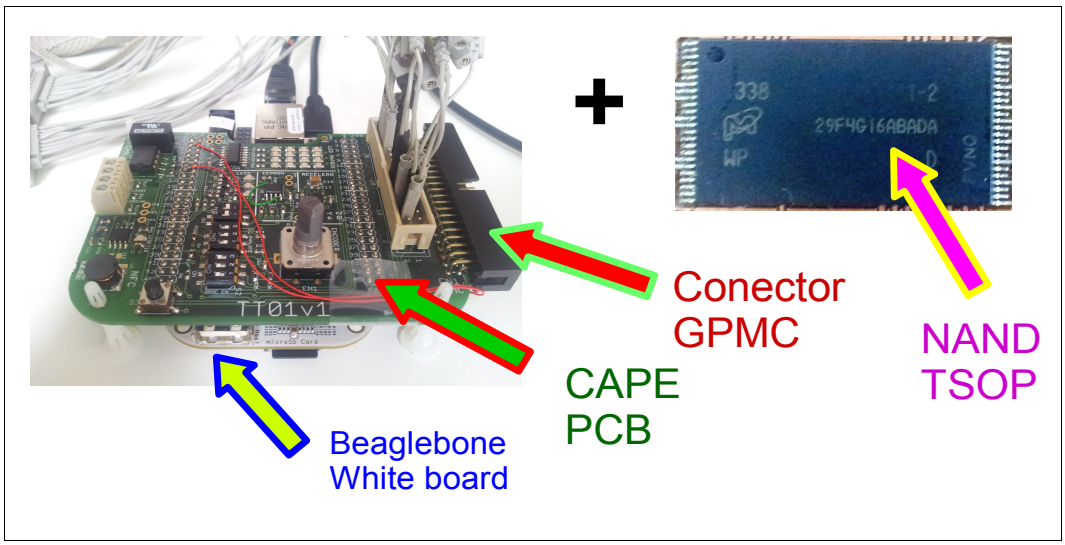


Fig. 1: *Beaglebone y componente periférico a comunicar.*

Se dispone de un PC como herramienta de trabajo con una partición de Ubuntu instalada. Se modifica su Kernel para actualizar el compilador cruzado y adaptar los ficheros generados de la compilación al nuevo sistema embebido de la beaglebone white Angstromg. El compilador cruzado instalado en un arquitectura de PC se emplea para generar los archivos binarios necesarios a ejecutar en la arquitectura AM335xx. Es decir, se escribe código fuente en C, se guardan con la extensión .c en cualquier directorio del sistema, se compilan y generan en el computador personal (a través de su makefile que habrá que adaptar también) para ser interpretados y ejecutados en la placa, dispositivo externo e independiente del ordenador. En otras palabras, aunque se compilen en el ordenador, las sus apps de la beaglebone no se pueden ejecutar en el PC.

Así pues, el entorno de trabajo para desarrollo del apartado de software es un PC con Ubuntu 64 bits versión 14.04 para generar archivos, establecer conexión con la placa y comunicarse con ella para acceder a su sistema de ficheros, etc. Entre otras cosas el PC brindará la interfaz gráfica para visualizar el terminal de acceso al sistema de ficheros de la placa.

En general, cuando una aplicación de usuario demanda una operación que requiere un acceso a un periférico es el controlador quién gestiona la orden y la traduce en operaciones de accesos al hardware. Este toma el dato requerido por la aplicación y se lo devuelve a la aplicación o lo conduce

hasta el hardware en función de si se trata de lecturas o escrituras. Por tanto, se precisa del diseño de una app que active estas peticiones al driver y active las operaciones pertinentes para las que se ha programado. Dado que trabajar a nivel de kernel implica escribir valores o leer bits en registros, el código fuente será implementado en C incluyendo librerías y cabeceras.

Ya se ha comentado anteriormente que la configuración de accesos NAND es mucho más compleja y laboriosa que un acceso NOR, además es un tipo de memoria proclibe a fallos de operación por bloques defectuosos. El driver de accesos NAND tiene unas configuraciones mínimas obligatorias. El resto son opcionales, pero optimizan y mejoran la calidad de los accesos.

El presente trabajo requiere desarrollo de software y hardware. El objetivo principal es obtener una interfaz integrada funcional para poder realizar futuros desarrollos más complejos, avanzados y sofisticados que pueden optimizar los accesos o permitir jugar con otras funciones. Por lo tanto, se realizará una versión del driver bastante básica. El mero intento de tratar de gestionar el control de errores puede dar lugar a un trabajo de años de programación para un profesional sin experiencia suficiente. Los profesionales acostumbran a trabajar a niveles superiores de desarrollo dejando este nivel de abstracción a sus proveedores de librerías.

Sin tener en cuenta el hardware a diseñar, el software se puede realizar y comprobar como bloque único. Esto permite trabajar en paralelo con el desarrollo de la placa de circuito impreso sin que interfieran etre sí. Ambos son bloques independientes hasta el momento de testar el comportamiento del driver mediante la observación del comportamiento de sus señales con el analizador digital.

Observar las señales lanzando operaciones de acceso. Si no se cumple se procederá a reajustar los parámetros temporales. Una vez obtenidos los resultados esperados se procede a la integración de la CAPE de la Beaglebone, PCB con dispositivo de memoria NAND y software: conexión y ejecucicón de la app.

Se procederá a una petición de operaciones sencillas como lanzamiento de RESET, lectura de estado, lectura de ID a la memoria, etc las cuales verificarían la funcionalidad tanto del hardware como de la interfaz. Aunque estas indican peticiones de lectura en NAND todas las peticiones posibles se realizan mediante activación de operaciones de escritura de comandos previas. Por lo tanto, en caso de que la lectura otorgue el resultado previsto a la aplicación ejecutada en Ubuntu embebdido puede darse por cumplido el objetivo principal. Implica que las señales de escritura y lectura están cumpliendo con lo establecido, llegando sin deteriorarse a los pines del chip de memoria. En caso contrario habría que ajustar los valores de los componentes smd empleados Rs de pull up, Rs jumper, Cs de desacoplo y de filtrado de armónicos, etc.

1.3.- Trabajo realizado anteriormente:

Como se ha citado anteriomente se han seguido una serie de pasos sugeridos por un trabajo anterior para la instalación del nuevo OS. Se proporciona una imagen del nuevo sistema como parte del material necesario para obtener una migración

- BeagleBoneUbuntu4GB_V05.rar: imagen del sistema operativo a instalar en la BB, en concreto en la flash card. Requerirá alguna herramienta que permita grabar la imagen en SD card.

Los desarrollos anteriores sobre la realización de un módulo CAPE para la asignatura y en concreto, la comunicación a través de su bus GPMC, pueden encontrarse en el apartado correspondiente del trabajo "*Design of a dedicated cape board for an embedded system lab course usingBeagleBone*" de Raúl Pérez López para Angstrom OS.

La versión original del driver entregada es un template con la estructura de un driver típico para Linux que accede al registro ID del uP, realizado para la asignatura. Los desarrollos se van a basar en él para las posteriores versiones: NAND. El material proporcionado para cargar el driver en el sistema consta de un set de ficheros .c y .h distribuidos del siguiente modo:

- carpeta lm_11 → directorio principal
  - carpeta application
    - fichero main.c → aplicación
    - fichero makefile para la app
  - carpeta module
    - fichero module.c → driver para linux
    - fichero makefile para el driver
    - carpeta de includes
      - ficheros.h → varios ficheros cabecera: algunas modificaciones leves van a ser necesarias en estas cabeceras. Se indicarán en el apartado de metodología.

3 elementos son imprescindibles para desarrollar la interfaz: un sistema embebido, una memoria flash NAND y un host para crear una red local y acceder remotamente al sistema embebido.

1.4.- Sistemas embebidos:

Los sistemas embebidos son sistemas de computación altamente integrados que ocupan poco espacio, bajo coste y tienen un consumo muy reducido. Dichos sistemas suelen estar diseñados para una o varias aplicaciones muy específicas.

Una característica de todo sistema embebido es que la mayoría de los componentes se encuentran incluidos en una placa y muchas veces los dispositivos resultantes no tienen el aspecto de lo que se puede considerar la típica computadora.

Un ordenador de propósito general está diseñado para cubrir un amplio rango de necesidades. Mientras que un sistemas embebido se diseña para cubrir necesidades específicas, pudiendo dar una respuesta en tiempo real. Su núcleo puede ser programado en ensamblador y C/C++. Si los requisitos temporales no son exigentes ni críticos pueden incluso emplearse lenguajes como Java.

Los sistemas embedded juegan cada día un papel más importante en los dispositivos electrónicos que nos rodean cotidianamente, sobretodo con la incipiente irrupción del internet de las cosas (IoT). Sistemas embedded los hay de varios niveles de complejidad. Desde la electrónica que gobierna el funcionamiento de un enrutador, reproductor DVD, hasta los de más alta complejidad como por ejemplo un smartphone debido a la gran cantidad de funciones y prestaciones que aportan. Típicamente los procesadores de un sistema embebido más complejo suelen venir acompañados de múltiples periféricos integrados en el propio encapsulado, sobretodo memoria. Se les denomina microcontroladores. Son empleados para aplicaciones de propósitos específico en muchos sistemas embebidos.

En términos de arquitectura lo que diferencia un ordenador de sobremesa o portátil y un sistema embebido es la arquitectura del procesador CISC vs microprocesador ARM RISC. Los CISC pertenecen a la primera corriente de construcción de procesadores, antes del desarrollo de los RISC. Los microprocesadores CISC tienen un conjunto de instrucciones que se caracteriza por ser muy amplio y permitir operaciones complejas. Este tipo de arquitectura dificulta el paralelismo entre instrucciones, por lo que, en la actualidad, la mayoría de los sistemas CISC de alto rendimiento implementan un sistema que convierte dichas instrucciones complejas en varias instrucciones simples del tipo RISC. Toda la familia Intel x86 o AMD x86-64 usada en la mayoría de las Computadoras Personales actuales son CISC. ARM es una arquitectura de set de instrucciones de 32b y 64b más ampliamente utilizado en la actualidad por la mayoría de procesadores y microcontroladores para mejorar su rendimiento. Necesitan una cantidad menor de transistores que los procreadores x86 CISC lo que lleva a una reducción de costes, calor y consumo. Fue concebida por el fabricante Acorn Business Computers (ABC) que cobra por licencia de uso en cada procesador que se utiliza esta arquitectura. CORTEX es una de las familias de procesadores ARM de alto rendimiento.

El área de los minicomputadores ha sido una de las áreas en las que la competencia ha dado respuesta más rápido desde su irrupción. El mercado de ordenadores de mínimo coste ha estado rápido presentando alternativas.

Puede dársele un uso en la robótica, automatización domótica, centro multimedia doméstico, computador de propósito general para navegación web, herramienta escolar, gaming, trabajo de oficina, dispositivo de prototipado para hardware, estación de desarrollo de software, entre muchas.

Arduino, BEAGLEBONE, Raspberry forman parte de la nueva generación de minicomputadores basados en procesadores CORTEX. Son placas de desarrollo conocidas como open hardware que se utiliza con fines educativos e industriales. Estas placas realzan y potencian las ventajas del open-hardware y software de código abierto ya que permiten realizar y mejorar gran variedad de funcionalidades.

BeagleBone es la placa de Beagleboard de bajo costo enfocada una alta expansibilidad usando un procesador ARM Cortex A8 low cost Sitara AM335x de Texas Instruments. La versión de Beaglebone empleada durante el proyecto es la Revision A6B cuyo procesador es concretamente un AM3358ZCZ72.

1.5.- Memorias flash; tecnología NOR vs NAND en sistemas embebidos

La memoria flash es un tipo de dispositivo no volátil (no es necesario que esté conectado a la corriente para que siga almacenando la información). El inventor del tipo de memoria flash fue Fujio Masuoka en 1984. Es una evolución de las memorias EEPROM (tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente), pero con una gran mejora tanto en el aspecto económico como en el físico, ya que son más pequeñas y por lo tanto fáciles de transportar, borrar y programar. Las EEPROM necesitan un dispositivo especial llamado lector de PROM. Sin embargo flash permite hacerlo desde el ordenador.

Además la velocidad es mucho mayor con respecto a las otras memorias existentes debido a su capacidad de borrar y escribir en una misma operación.

En la actualidad el desarrollo de las memorias flash es mucho más rápido en comparación con otras memorias. Esto es posible gracias al uso personal que se dan a estas memorias que permite seguir invirtiendo en el desarrollo de tecnología para las memorias flash. La memoria Flash NAND se inventó después de la memoria Flash NOR.

Flash NOR es excelente en aplicaciones donde los datos se recuperan o se escriben de manera aleatoria. Por ello NOR se encuentra más frecuentemente integrada en sistemas para almacenar firmware o el programa de la BIOS de arranque de un computador.

Cada acceso a/desde NAND ha de ser precedido de varios ciclos de escritura indicando la operación y la dirección cosa que NOR no requiere. Cada fabricante dispone de un set de comandos para su modelo NAND. Las operaciones son activadas por comandos y señales de control. Mientras que a NOR le basta con la señalización de control para interpretar e iniciar sus accesos. El tipo NOR permite una escritura más lenta que NAND, pero procesa muy rápido las rutas de acceso aleatorias. Esto hace que NOR sea una solución óptima para la ejecución y almacenamiento de instrucciones de programa, mientras que NAND es más indicado para el almacenamiento masivo de datos.

NAND puede almacenar más datos en un espacio de silicio más pequeño, lo que ahorra el coste por bit. En el pasado, cuando el almacenamiento de datos era más bajo, NOR tuvo mayor influencia en el mercado. Tradicionalmente los sistemas embebidos han estado empleando NOR para memoria no volátil.

Las eficiencias NAND son debidas en parte al menor numero de contactos metálicos requeridos. El tamaño es considerablemente menor, a razón de 4F^2 por celda NAND y 10F^2 (F=WxL) por celda NOR, ya que requieren contactos metálicos separados por cada celda.

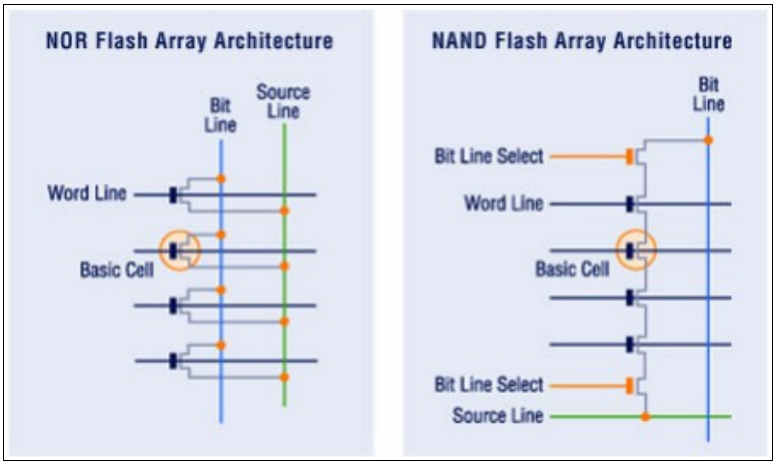


Fig. 2: *Comparación de arquitecturas entre ambas tecnologías*

Más diferencias a nivel estructural es la cuenta de pines; NOR Flash necesita 44 I/O pines para 16-bit mientras que NAND emplea sólo 24 pins.

NAND activa su petición de operación mediante envío de comandos, direcciones y transacciones en dos sentidos de datos por el mismo bus de datos. Esto permite migrar hacia dispositivos de diferentes densidades y ancho de datos sin necesidad de rectificar el resto del hardware de la PCB en la que se integra.

NAND Flash es la más apta para archivos de datos y aplicaciones de datos secuenciales. NOR Flash es una tecnología más adecuada para accesos random. NAND proporciona operaciones de borrado y escritura más rápido. Sin embargo NOR es la más eficiente para el acceso aleatorio y escrituras de un byte. Gracias a ello, a NOR posee la funcionalidad XiP (execute-in-place) requisito indispensable para arrancar el sistema desde ese chip de memoria. Aunque cabe decir que cada vez hay más procesadores que incoporan una interfaz directa hacia la NAND para poder iniciar el boot desde la memoria NAND. Pero sin duda, la principal desventaja de esta memoria es la lentitud de sus accesos random, motivo por el cual se ha desaconsedo tradicionalmente su empleo para dicho modo de operaciones. Por contra NOR tiene tiempos de escritura y borrado mucho más lentos.

Si bien algunas memorias NOR pueden leer mientras escriben, NAND no puede. Sólo es posible realizarlo a nivel de sistema usando métodos de sombreado (shadowing). Es un método que no es desconocido ni novedoso, ya que es el mismo que se viene empleando desde hace años para cargar la BIOS desde ROMS más lentas hacia RAMs más rápidas en el campo de los computadores. Esta técnica permitiría realizar accesos aleatorios (a nivel de sistema ocultando datos a la RAM y requiere RAM adicional).

Otra característica de las memorias NAND es que son más susceptibles a los fallos. Por ello la mayoría son compatibes con diferentes técnicas y algoritmos que permiten detectar y bloquear el acceso a bloques corruptos. Es recomendable trabajar con estas técnicas habilitadas para aumentar garantías frente a los fallos. De no activar ningún mecanismo aumentaría el riesgo de accesos fallidos. Otros algoritmos balancean el uso desigual en nº de veces que unas zonas son accedidas y otras no, para que todas las zonas tengan el mismo nº de usos en promedio. Tienen un ciclo de vida mayor que las NOR, hasta diez órdenes más.

No obstante el uso de NOR está generalizada en la industria porque ofrece una interfaz más sencilla y además puede ejecutar código. Por su arquitectura consiste en un buen arreglo hasta los pocos MB. Ofrece un alto rendimiento para las lecturas pero mucho peores tiempos para la escritura. Óptimo para las aplicaciones con accesos sólo lectura. Se desaconseja para almacenamiento de datos. Las evoluciones NAND cada vez suponen más una buena solución para combinar ambas cosas: reservar una zona para programa y otra para datos con mayor capacidades que NOR a precios más económicos.

	NAND	NOR
Advantages	Fast PROGRAMs	Random access
	Fast ERASEs	Byte PROGRAMs possible
Disadvantages	Slow random access	Slow PROGRAMs
	Byte PROGRAMs difficult	Slow ERASEs
Applications	File (disk) applications	Replacement of EPROM
	Voice, data, video recorder	Execute directly from nonvolatile memory
	Any large sequential data	

Fig. 3: *Cuadro resumen de NOR vs NAND.*

Resumiendo, para NAND:

- La interfaz (serie) y diseño software del control es mucho más complejo que NOR.
- Mapeo I/O de NAND: direccionable por bus de datos.
  - NOR: mapeo completo de sus direcciones.
- Los accesos random implican una técnica llamada shadowing y unos tiempos mucho mayores.
- Accesos secuenciales y en ráfagas: borrados por bloques, escrituras y lecturas por páginas.
- NAND es legible y escribible. Aunque para escribir, primero hay que borrar (sólo escribe '0's).
  - NOR tiene una escritura menos eficiente: aplicaciones mayoritariamente sólo lectura.
- Óptima para almacenamiento de datos de alta densidad y capturas de datos que requieren mejores tiempos.
- Más económica y menos espaciosa (mayor densidad).
- Los tiempos de escritura son menores.

La siguiente gráfica ilustra de forma concisa los principales campos de aplicación óptimos para estas 2 tecnologás.

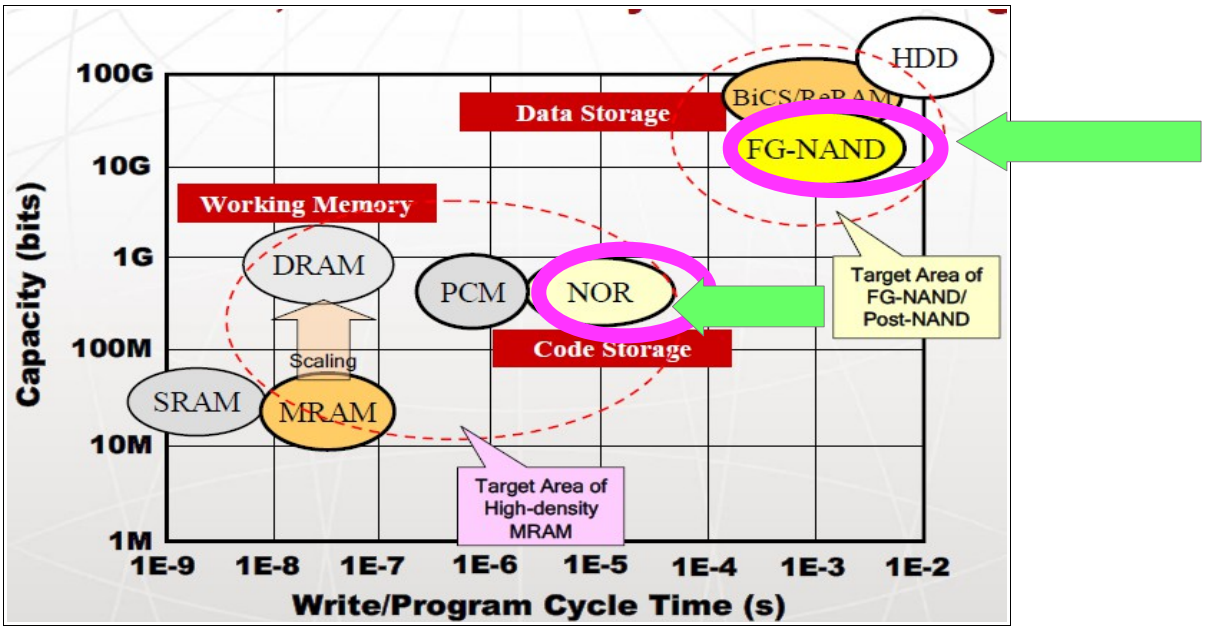


Fig. 4: *Relación de capacidades y tiempos en función de la tecnología.*

1.6.- Bus GPMC

GPMC (controladores de memoria y de uso general) es un controlador de memoria unificada para conectar dispositivos de memoria externos como NAND, NOR, SRAM asíncrona etc.

Tiene una máxima capacidad de direccionamiento de 512MB que puede ser dividido en siete chip de memoria controlados por CS con el tamaño del banco programable y de dirección base de 16 MB, 32MB, 64MB, 128MB o 256MB independientemente.

Cada CS tiene unos parámetros tiempos de hold y setup que pueden ser programados asociados a las señales de control. Deben fijarse en los registros del bus, basándose en los requisitos de los parámetros de la memoria con un factor de escala temporal del reloj de la interfaz slow L3. Cuyo reloj FCLK proporciona una velocidad máxima de 100Mhz (10ns por ciclo).

Mediante la programación de sus registros, una aplicación puede ser capaz de acceder al mencionado tipo de dispositivo a través del bus GPMC. GPMC posee el motor de código de corrección de errores (ECC). Soporta distintos algoritmos de ECC como código de Hamming, BCH 4, 8 y 16 bits. Tiene otras dos funciones avanzadas opcionales que aumentan la velocidad de las operaciones: Prefetch y write posting engine. Puede ser utilizado leer o escribir en modo buffer. También GPMC proporciona acceso DMA que se puede utilizar junto con el prefetch para aumentar el rendimiento NAND para leer / escribir, entre otras características que pueden hallarse en el capítulo introductorio del manual de referencia de la CPU.

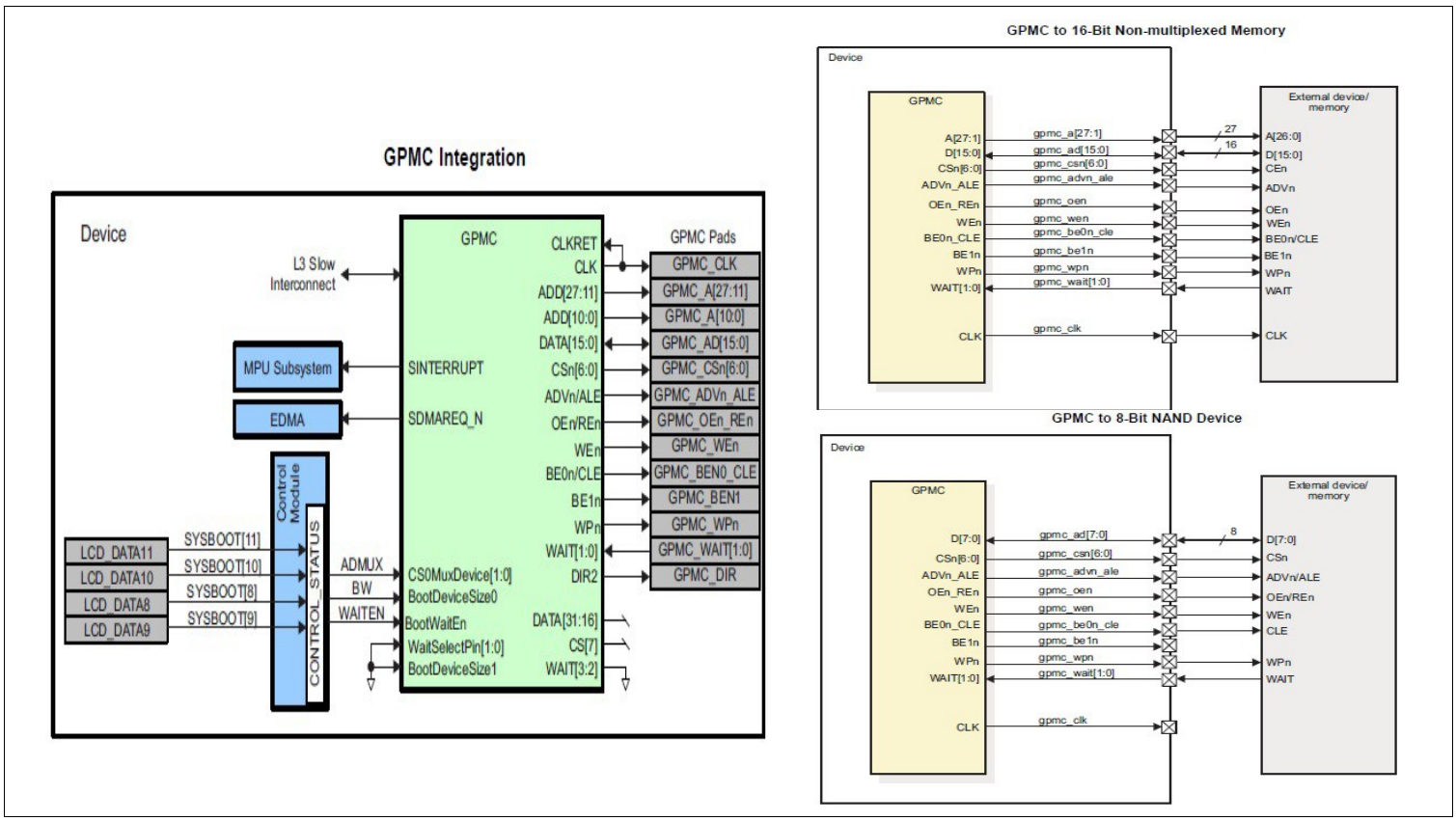


Fig. 5: *a) Integración del driver GPMC en el uP, b) interfaz GPMC – memoria sync, c) GPMC – NAND*

A partir de los campos de bits de configuración programados y almacenados en los registros de GPMC, el host GPMC es capaz de generar todas las señales de control de temporización según el dispositivo y tipo de acceso necesario. Dada la decodificación de selección de chip y sus registros de configuración asociados, GPMC selecciona el tipo de dispositivo de control de señales de temporización apropiada.

Signal	Type	Description
GPMC_A[27:0]	O	Address outputs
GPMC_AD[15:0]	I/O	Data[15:0] in non-muxed mode. A[16:1], D[15:0] in AD-muxed mode. A[27:17], A[16:1], D[15:0] in AAD-muxed mode.
GPMC_CSn[6:0]	O	Chip selects (active low)
GPMC_CLK	O(1)	Synchronous mode clock
GPMC_ADVn_ALE	O	Address Valid or Address Latch Enable depending if NOR or NAND protocol memories are selected.
GPMC_OEn_REn	O	Output Enable (active low). Also used as Read Enable (active low) for NAND protocol memories
GPMC_WEn	O	Write Enable (active low)
GPMC_BE0n_CLE	O	Lower Byte Enable (active low). Also used as Command Latch Enable for NAND protocol memories
GPMC_BE1n	O	Upper Byte Enable (active low)
GPMC_WPn	O	Write Protect (active low)
GPMC_WAIT[1:0]	I	External wait signal for NOR and NAND protocol memories.
GPMC_DIR	O	GPMC.D[15:0] signal direction control Low during transmit (for write access: data OUT from GPMC to memory) High during receive (for read access: data IN from memory to GPMC)

Fig. 6: *Señales de control del bus GPMC*

2.- Estado del arte y motivaciones para la elección de la tecnología empleada.

2.1.- Minicomputers

Estas plataformas de desarrollo están detrás de la ingeniería encontrada en los dispositivos más innovadores de la actualidad, sobretodo el IoT y los dispositivos "wearables". Son herramientas fácilmente localizable en todo laboratorio como origen de la creación de los prototipos. No hay que olvidar la importante labor de este tipo de plataformas en los entornos docentes. Por lo tanto se presenta una lista de los modelos más conocidos y empleados en la industria :

2.1.1.-Intel Edison

Placa con SoC de doble núcleo de Intel Atom que tiene integrado Wi-Fi, Bluetooth LE y un conector de 70 pines que permite a los usuarios conectar una variedad de periféricos. Soporta Yocto Linux, Arduino, Python, Node.js y Wolfram, así que es tan flexible como potente. Tiene un bajo consumo de energía y su diseño en miniatura convierten a la Intel Edison en una excelente opción para los diseños sensibles al tamaño e integrados. Se realiza con compatibilidad con Arduino. Existen otras placas que la proveen de mayor funcionalidad y se emplean para conectar con otras placas.

CLK	I/O	VDD	Memoria	Core	PWM	Interfaces
500 Mhz x2 Core	70	I: 3.15V	RAM: 1GB	ATOM UP: ARM1176JZF-S	4 (I/O)	USB
		Max: 4.5V		32b		I2C
		O: 1.8V				Ethernet
						uSD
						GPIOs
						Wi-fi
						SPI
						UART
						BT 4.1
						MMC

Tabla 1: resumen características Intel Edison

2.1.2.-Raspberry Pi

Las aplicaciones en el Raspberry Pi pueden ser fácilmente construidas y probadas usando Python, que es el lenguaje de programación principal destinada a este tablero. Otros lenguajes como C, C ++, Java y Perl también son factibles para desarrollar aplicaciones en el Raspberry Pi. Gracias a su slot para microSD, la Raspberry Pi B+puede manejar cualquier sistema operativo desde una distribución Linux hasta Windows 10. Con 4 puertos USB, salida HDMI, Micro USB y un RJ45 Ethernet 10/100, herramienta totalmente abierta. Familia de Rasp Pi por orden de lanzamiento: Raspberry Pi 1 Modelo A, Modelo B, Modelo B+, modelo 2, Modelo 3.

Rasp PI B+:

Es el modelo de la gama que representa el primer cambio significante desde el primer dispositivo. Ha aumentado la cantidad de memoria RAM a 1 GB, aunque se mantiene el ARM11 de coma flotante. Sí se ha mejorado en cambio el consumo, gasta entre 0.5 y 1 W menos de energía. También mejora la calidad del audio al tener fuente propia. B+ also incluye el mismo puerto para el módulo de cámara de las versiones anteriores y es compatible con los mismos sistemas opeativos (Raspian, OpenELEC, OSMC, etc.).

CLK	I/O	VDD	IDC	Memoria	Core	PWM	Interfaces
700 Mhz	40	5V	5v: 600mA	RAM: 512 MB	BCM2835 UP: ARM1176JZF-S	3 (I/O)	4 USB
		3,3V	I/O3.3V:50mA		32b		I2C
							Ethernet
							uSD
							CSI(Cámara)
							DSI(Display)
							SPI
							UART
							HDMI
							Full HD

Tabla 2: *resumen características Rasp Pi B+*

Rasp PI 3:

Los desarrolladores aseguran que el nuevo procesador es diez veces más potente que el de la Raspberry original y un 50% más rápida que la segunda Raspberry. Funciona a 1,2GHz y es de 64 bits. La memoria RAM se mantiene en 1GB. Es la evolución más importante en potencia y posibilidades, al incorporar conectividad Wi-Fi, Bluetooth low energy y un **nuevo procesador de 64 bits**con cuatro núcleos (ARM Cortex-A53). Compatibilidad completa con with Raspberry Pi 1 and 2. Because it has an ARMv7 processor, it can run the full range of ARM GNU/Linux distributions, including Snappy Ubuntu Core, as well as Microsoft Windows 10.

CLK	I/O	VDD	IDC	Memoria	Core	Interfaces
1.2 Ghz 4 Core	40	5V	800mA	RAM: 1GB	BCM2837 uP: ARM-A53	4 USB
				DMA	64b	I2C
						Ethernet
						Wi-fi
						uSD
						CSI(Cámara)
						DSI(Display)
						SPI
						UART
						HDMI

Tabla 3: *resumen características Rasp Pi 3*

2.1.3.-ODROID-C2

Una de las placas de desarrollo de 64b más económicas en el sector de los ARM. Es una solución asequible que mejora en algunos aspectos a Pi 3. Ofrecen un equipo que es más rápido que Raspberry Pi 3, además de doblarlo en memoria. Está un paso por detrás en conectividad inalámbrica: no cuenta con soporte WiFi, tampoco Bluetooth. También dispone cuatro puertos USB 2.0, un receptor para infrarrojos, y una salida HDMI 2.0 que permite sacar una salida 4K/60Hz, con soporte H.265/H.264.

El chipset que lleva la placa C2 es un Amlogic S905, que tiene un procesador de cuatro núcleos funcionando a 2GHz, con un tipo de núcleo ARM A53, también.

Algunos de los sistemas operativos más modernos que pueden cargarse en ODROID-C2 son Ubuntu, Android, ARCHLinux, Debian, con miles de paquetes open source disponibles.

Los 40 pines GPIO en un ODROID-C2 son una forma de interactuar con los dispositivos físicos como los botones y los LED utilizando un controlador sencillo de Linux. Para desarrollos C / C ++ o Python, hay la biblioteca WiringPi es muy útil para interactuar con los pines.

CLK	I/O	VDD	IDC	Memoria	Core	Interfaces
2 Ghz 4 Core	GPIO:40	5V	500mA	RAM: 2 GB	Amlogic S905, uP: ARM-A53	4 USB
	I2S: 7				64b	I2C
						Ethernet
						uSD
						Cámara USB 720 p
						Recept infrarojos
						SPI
						UART
						HDMI (audio & video)
						PWM

Tabla 4: *resumen características ODROID-C2*

2.1.4.-Arduino

Arduino es una compañía dehardware libre, la cual desarrolla placas de desarrollo que integran un microcontrolador y un entorno de desarrollo (IDE) basado en el entorno de Processing y un lenguaje de programación basado en Wiring, así como en el bootloader que es ejecutado en la placa. Incluye soporte a una versión simplificada de C y C++.

El hardware consiste en una placa de circuito impreso con un microcontrolador, usualmente Atmel AVR, puertos digitales y analógicos de entrada/salida; 4 los cuales pueden conectarse a placas de expansión (shields) que expanden las características de funcionamiento de la placa arduino.

Arduino ofrece una gran variedad de productos entre placas, shields y otros complementos. Las frecuencias de reloj entre las que oscilan sus modelos son los 8 Mhz y los 86Mhz, Arduino Due. La gama de placas Arduino son: UNO, 101, PRO , MEGA, ZERO, DUE, YÚN y 5 modelos Wearables más.

Arduino Uno:

Arduino Uno SMD R3 es una placa basada en el uC Atmega328. Con 14 input/output pins digitales, 6 inputs analógicos, un oscilador de 16 MHz, con conexión USB, un jack para alimentación, una cabecera ICSP (programación en circuito) y botón reset. Contiene todos los componentes para soportar el uC, se conecta a la computadora vía USB o se alimenta por medio del adaptador AC/DC.

Pinout: incluye pines SDA y SCL para la comunicación TWI próxima al pin AREF y otros dos más cerca del pin RESET, IOREF permite a los shields adaptarse al voltaje provisto desde la placa.

CLK	I/O	VDD	IDC x pin	Memoria	Core	PWM	estándars
16Mhz	Dig: 14	5V	I/O: 40mA	SRAM: 2KB	uC: ATmega328	6 (I/O)	USB
	Anal: 6	7/12V	3.3v: 50mA	EEPROM: 1KB			I2C
		Max: 20V		Fjash: 32KB, 0,5KB para boot			ICSP

Tabla 5: *resumen características ArduinoUno*

Arduino Yun:

Arduino Yun es un miembro de una nueva línea innovadora de productos WiFi que combinan el poder de Linux con la facilidad de uso de Arduino. El Arduino Yun es la combinación de un clásico Arduino Leonardo (basado en el procesador Atmega32U4) con un sistema de WiFi on-chip funcionando con OpenWrt-Yun (una distribución GNU / Linux basada en OpenWrt). Es una máquina Linux embebida directamente en la PCB de la Arduino Leonardo. Pero se conectan los dos para que a partir de Arduino sea muy fácil de ejecutar comandos en el lado de Linux y utilizarlo como una interfaz Ethernet y Wi-Fi.

CLK	I/O	VDD	IDC x pin	Memoria	Core	PWM	estándars
16Mhz	Dig: 20	5V	I/O: 40mA	SRAM: 2.5 KB	uC: Atmega32u4	7 (I/O)	USB
	Anal: 12	Max: 5V	3.3v: 50mA	EEPROM: 1KB			I2C
				Flash: 32KB, 4KB para boot			Ethernet
							uSD

Tabla 6: *resumen características Arduino Yun*

Arduino Due:

El Arduino Due es una placa electronica basada en la CPU Atmel SAM3X8E ARM Cortex-M3. Es la primera placa Arduino basada en un microcontrolador núcleo ARM de 32 bits. Cuenta con 54 pines input/output digitales (de los cuales 12 se pueden utilizar como salidas PWM), 12 entradas analógicas, 4 UARTs (puertos serie de hardware), un reloj de 84 MHz, una conexión capaz USB OTG, 2 DAC (de digital a analógico) , 2 TWI, un conector de alimentación, una cabecera de SPI, un encabezado JTAG, un botón de reinicio y un botón de borrado.

CLK	I/O	VDD	IDC x pin	Memoria	Core	PWM	estándars
86Mhz	Dig: 54	3.3V	I/O: 130 mA	SRAM: 96 KB	uC: AT91SAM3X8E, ARM M3	12 (I/O)	USB
	Anal: 12	7-12V	3.3v, 5v: 800mA	Flash: 512 KB	32b		I2C
		Max: 6 -16V		DMA			Ethernet
							uSD
							SPI
							UART

Tabla 7:*resumen características Arduino Due*

2.1.5.-Beaglebone White

Es un mini ordenador Linux del tamaño de una tarjeta de crédito, con un coste de $ 89, que se conecta a Internet y ejecuta el software como Android 4.0 y Ubuntu. Proporciona gran cantidad de E / S dos conectores de expansión de 46 pines y la potencia de procesamiento para el análisis en tiempo real proporcionada por un procesador Sitara ARM Cortex A8 de 720MHz AM335x, 256 MB de RAM, Ethernet en el chip, una ranura microSD, un puerto USB host y puerto del dispositivo multifuncional

que incluye control en serie de bajo nivel y conexiones JTAG de depuración de hardware, por lo que no se requiere ningún emulador JTAG.

Un número de estas CAPEs se han publicado recientemente.

CLK	I/O	VDD	IDC x pin	Memoria	Core	PWM	estándars
720Mhz	2x46	I/O: 5V	5V: 500mA	RAM: 256MB	uP: ARM A8	8 (I/O)	USB
		O: 3.3V			32b		4xUART
							Ethernet
							GPMC
							uSD
							Graph 3D
							CAN
							MMC
							I2C
							SPI
							LCD
							4xLEDs

Tabla 8: *resumen características Beaglebone White*

2.1.6. - BeagleBone Black

El modelo *black* de BEAGLEBONE es un tablero bajo costo de gran alcance que es el sucesor del modelo *white* de BEAGLEBONE. Se proporciona un procesador más rápido, que es un ARM Cortex A8 a 1GHz. La memoria RAM también se ha mejorado, proporcionando 512MB DDR3. Una mejora importante en comparación con el modelo *white* es que incluye una interfaz HDMI micro, lo que sumado a su poder hace esta placa ideal para aplicaciones multimedia. Esta placa puede obtenerse por 60 € aproximadamente.

2.1.7- Modelo empleado

La BeagleBone viene dotada de dos conectores (headers) de expansión P9 y P8, usados frecuentemente para conectar las CAPEs. Estos conectores son dos columnas de 46 pins dispuestos en columnas de 2x23 pines cada una. De forma que ofrece un set de 65 GPIOs y otras interfaces.

Una CAPE PCB es el nombre de una placa de expansión para BB. Una CAPE es el mismo concepto que los SHIELDs para Arduino. Estas placas de expansión pueden ser adquiridas comercialmente o diseñadas de forma autónoma e independiente por un usuario. Una CAPE puede tener una variedad de periféricos controlables por el procesador. O bien, pueden implementar la interfaz de un único periférico.

Por otro lado una CAPE requiere una EEPROM con información concreta para que el sistema reconozca y cargar la correcta descripción del hardware. La EEPROM que está en la cape almacena diferentes parámetros de información, como un usuario, número de serie, número de parte, la información de revisión y otros. La única información que el gestor de la cape requiere es el número identificador y la revisión.

La arquitectura del núcleo es compatible con sistemas operativos embebidos como Debian, Ubuntu, Android o Arch.

2.1.7.1.- Lab Cape TT01v1

Tal como se adelanté en otro apartado esta cape fue desarrollada en un proyecto anterior para conectarla a la BB y ser empleada en la asignatura. Fue diseñada para trabajar usando un cape manager bajo el kernel 3.8. En el arranque, el kernel carga el fichero DTB, **am335x-bone.dtb**. Un Device Tree Blob (DTB) es un fichero binario describiendo el hardware del sistema completo. El archivo será cargado por el bootloader en el momento boot y analizado por el kernel. Mientras un DTB se refiere a ser analizado cada momento de arranque, las CAPEs no son estáticas. Esto lleva a que diferentes CAPEs deben ser conectadas cada vez que la placa arranca y la única forma de reconocer qué tipo de CAPE está conectada será consultando la EEPROM en cada CAPE.

El archivo que puede cargar dinámicamente en el kernel en ejecución es cape-bone-TT01v1-00A0.dtbo y es gestionado por el CAPE manager. Más allá del 3.8 estos dos conceptos desaparecen. De otro modo añadir la descripción al Kernel parece que puede ser un proceso tedioso hasta que un

nuevo método más flexible aparezca, como todo indica que va a suceder con próximas ramas de kernel.

Un dtbo aplica los cambios a la representación interna del dispositivo en el kernel y activa los cambios que la acción implica. Habilitar y suprimir un nodo de dispositivo se traduce en cargar y eliminar un dispositivo.

La cape-bone-TT01v1 es un desarrollo hardware destinado a propósitos educativos. Esta Cape es compatible con ambas versiones de BeagleBone: BeagleBone White y Black. Esta Cape se acopla sobre la BB usando P8 y P9. Integra varios dispositivos e interfaces a través de los cuales pueden llevarse a cabo experimentos durante las sesiones de laboratorio. A continuación, una lista de interfaces que agrupa esta Cape:

- **Cape EEPROM**
- **Cape LEDs**
- **Cape push button**
- **Cape analog input**
- **Cape rotary encoder**
- **Cape NFC**
- **Cape CAN bus**

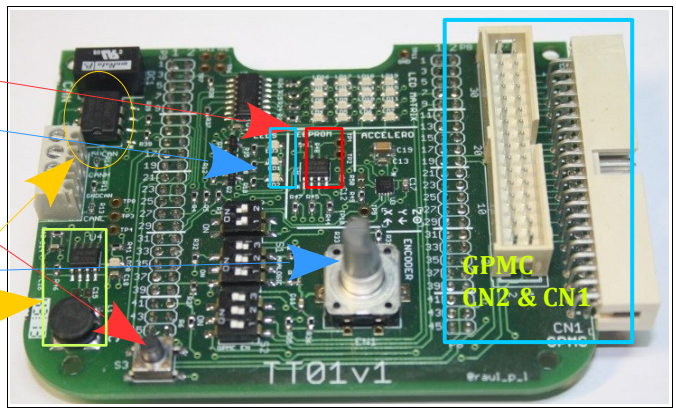


Fig. 7: *Ilustración de la CAPE-BONE-TT01V1*

2.1.7.2.- Cape GPMC interface

The General Purpose Memory Controller (GPMC) is a memory controller aimed to interface external memory devices:

- Asynchronous SRAM-like memories and ASIC devices
- Asynchronous, synchronous, and page mode burst NOR flash devices
- NAND Flash
- Pseudo-SRAM devices

Un set de señales GPMC disponibles procedentes desde los pines de P8 y P9 son llevados a las cabeceras CN1 and CN2 situados la Cape TT01 en un recuadro azul de la imagen precedente. Esto permite comunicación con unos tipos concretos de dispositivos de memoria externa. CN2 se implementa para permitir una conexión fácil con el analizador lógico y hacer lecturas del comportamiento de estas señales antes de conectar con el periférido de memoria. CN1 ha sido diseñado para conectar con un dispositivo, como una placa Altera DE2 basada en FPGA, por su puerto GPMC e incorpora otras señales como las del estándar SPI.

2.2.- Dispositivos Flash Nand

Hoy, con el gran incremento de la necesidad de capturar y almacenar más datos, en el mercado de consumo de dispositivos electrónicos, NAND ha superado de lejos a NOR, debido al aprovechamiento de su alta densidad (equivale a mayor nº de bits en menos espacio) y representar una solución de bajo coste para aplicaciones de altas prestaciones: menos consumo, menor tamaño, ligero, robusto, etc..

La demanda de bajo consumo y tendencia a la miniaturización de dispositivos ha llevado a NAND flash a convertirse en la opción de almacenamiento líder para una amplia gama de aplicaciones. Hasta los pocos GB pueden satisfacer con creces los requisitos de almacenamiento gráfico de un disco duro.

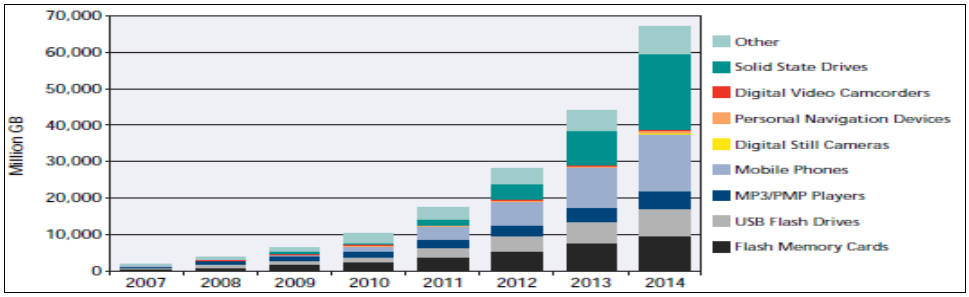


Fig. 8: *Evolución de la demanda de NAND Flash del mercado*

En la actualidad existen dos tipos de NAND en cuanto al factor de densidad; SLC y MLC. Cada celda equivale a 1 bit en los tipos SLC (dispositivos single level), pero existen modelos MLC (multilevel) de más alta densidad NANDs que aumentan su densidad almacenando más de un bit por celda.

Cada bloque tiene una media de 100000 ciclos de escritura y borrado de vida. Dos técnicas como el manejo de bad-blocks y wear leveling maximizarán sus ciclos de vida.

Una memoria de 2 GB (256MBytes) se divide en varios bloques, los cuales pueden alcanzar los 128KB. Un bloque es la unidad mínima borrable. La escritura sólo pude darse en un sentido, escribiendo 0. Por lo tanto es necesario un borrado previo dejando todos los bytes del bloque a 'FFh' (es lógica negada).

Una estructura típica para una matriz de 2 GB NAND puede constar de 2048 bloques, 64 páginas por bloque y 2112 bytes por página si su interfaz es de 8 bits. Si fuesen 16 bits podrían ser 1056 bytes. Existe una zona reservada (Spara Area) para escribir y guardar bits de las dos técnicas mencionadas anteriormente.

La siguiente figura muestra de una forma muy gráfica la estructura por bloques de una matriz NAND como la que ha sido descrita más arriba:

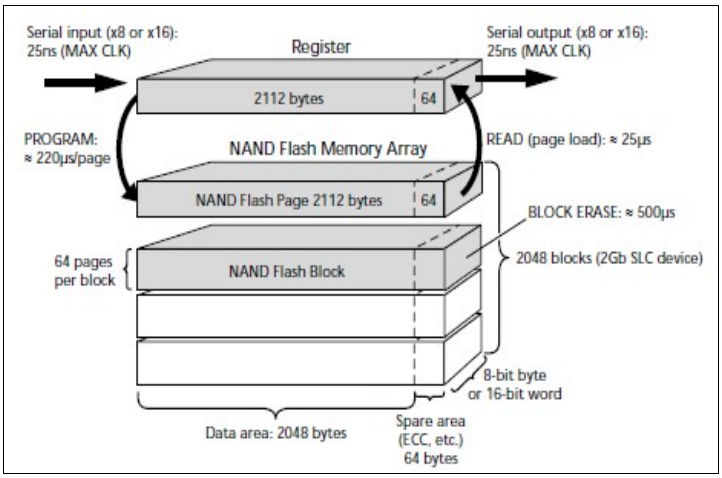


Fig. 9: *Organización de una matriz NAND de 2 GB*

Otro denominador común en la interfaz es la multiplexación de bus da datos para direccionamiento y petición de accesos. Teniendo en cuenta que cada ciclo de acceso equivale a un acceso de palabra de interfaz, una operación de lectura de página requiere una secuencia de: un ciclo de esritura del comando, una media de 4/5 ciclos de escritura con dirección de columna y dirección de fila (bloque y página), un ciclo de fin de comando y a continuación sus ciclos de lectura necesarios para capturar todas las palabras de la página con el bus en modo salida.

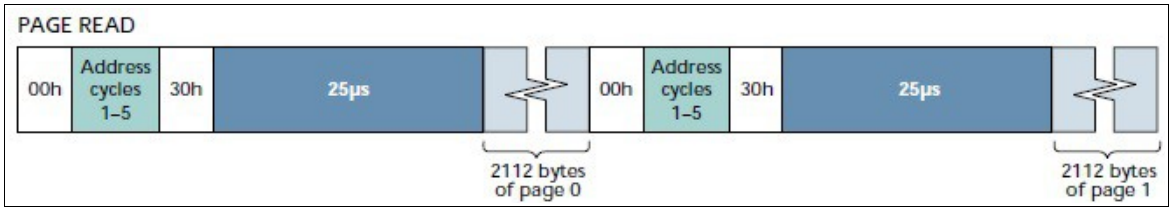


Fig.10: *Ejemplo de una operación de lectura de página; multiplexación en tiempo del bus I/O.*

En el direccionamiento típico NAND suelen usarse sólo 8 bits por ciclo, indpendientemente del ancho de su interfaz:

Cycle	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0
First	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0
Second	LOW	LOW	LOW	LOW	CA11	CA10	CA9	CA8
Third	BA7	BA6	PA5	PA4	PA3	PA2	PA1	PA0
Fourth	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8
Fifth	LOW	LOW	LOW	LOW	LOW	LOW	LOW	BA16

Notes: 1. Block address concatenated with page address = actual page address. CAx = column address; PAx = page address; BAx = block address. The page address and the block address, collectively, constitute the row address.

Tabla 9: *Partición de dirección completa en direcciones parciales en una memoria de 2GB*

Con esta introducción sobre la organización común en típica de una memoria NAND se analizarán los principales fabricantes del mercado y su surtido de modelos NAND. Se valorarán las apuestas más adaptadas a las necesidades de la interfaz a desarrollar y se seleccionará la mejor candidata.

2.2.1.- Soluciones para conectar una NAND a un procesador RISC

A la hora de conectar una memoria externa al procesador/controlador host existen dos posibilidades; una es la solución RAW NAND y la otra es eMMC. Esta última es una alternativa más cómoda que requiere una interfaz MMC.

Raw NAND requiere un controlador host que soporte una interfaz directa de un ARM contra un dispositivo NAND flash, a su vez las técnicas de gestión de bloques y wear laveling deben ser manejadas por software en el controlador. e•MMC por otro lado integra el controlador y el array NAND flash en su mismo encapsulado. El diseño de una interfaz con este tipo de dispositivos suprime gran parte del desarrollo software de control. Bastaría con un sencillo driver como software del host controlador. Actualmente es ofertado mayoritariamente en un único formato, BGA (pines de bolas en la base del chip).

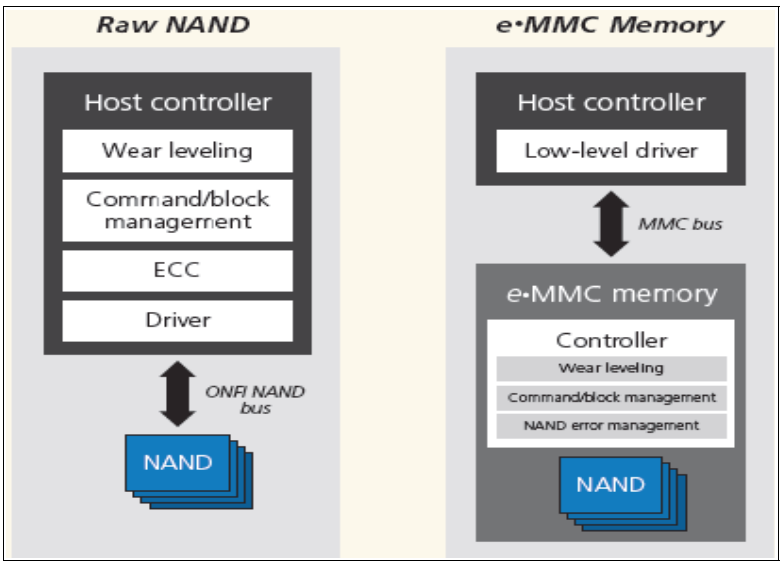


Fig. 11: *Comparativo entre interfaz Row y eMMC*

La primera opción puede parecer más cómoda, eficiente y menos costosa temporal y económicamente cuando se trata de controlar varios dispositivos NAND. Pero del otro modo se tiene la ventaja de personalizar la solución y crear un driver a medida.

No obstante, por ahora, la asignatura enfoca el acceso a memoria externa al bus GPMC en su docencia; además de que la CAPE desarrollada en trabajos previos dispone una interfaz para trabajar con este bus. Varios son los motivos por los que la solución eMMC se descarta:

- Para seguir la línea y trayectoria del material docente. Por el nivel de abstracción académica del que es objeto la asignatura, eMMC no encaja, ya que este encapsula la interfaz GPMC.
- Para interconectar la otra alternativa de interfaz debería modificarse la CAPE o bien realizar un nuevo diseño CAPE para controlador eMMC.
- Sobretodo el tipo de encapsulado en que se presenta este dispositivo que convierte en prohibitivo el proceso de soldadura para un primer prototipo.

El procesador AM3358ZCZ72 empleado tiene mapeados 512MB (2GB) dedicados a la interfaz GPMC. Puede direccionar un mínimo de 16MB y un máximo de 256MB por dispositivo de memoria conectado, denominado CS y hasta un máximo de 7 CS. Debido a que NAND no es una memoria mapeada en el espacio de memoria de la CPU, el acceso a la totalidad de la memoria está garantizado aunque se le asigne a la CPU el espacio mínimo de 16MB. Lo cual permitiría repartir y asociar el resto del espacio (16MB – 512MB) a otras memorias sí mapeadas en el espacio de memoria. Por tanto se ha centrado la búsqueda de un modelo en el rango de volúmenes entorno al 1 y a los 2 GB.

2.2.2.- Principales fabricantes y productos analizados

Esta estadística muestra los 6 principales fabricantes de memoria NAND Flash en todo el mundo y su cuota de mercado desde el primer trimestre de 2010 al cuarto trimestre de 2015. Ocupando Samsung el primer lugar, le sigue Toshiba experimentando una reducción de cuota en los últimos años que coincide con la irrupción de sanDisk, tercero en el ranking.

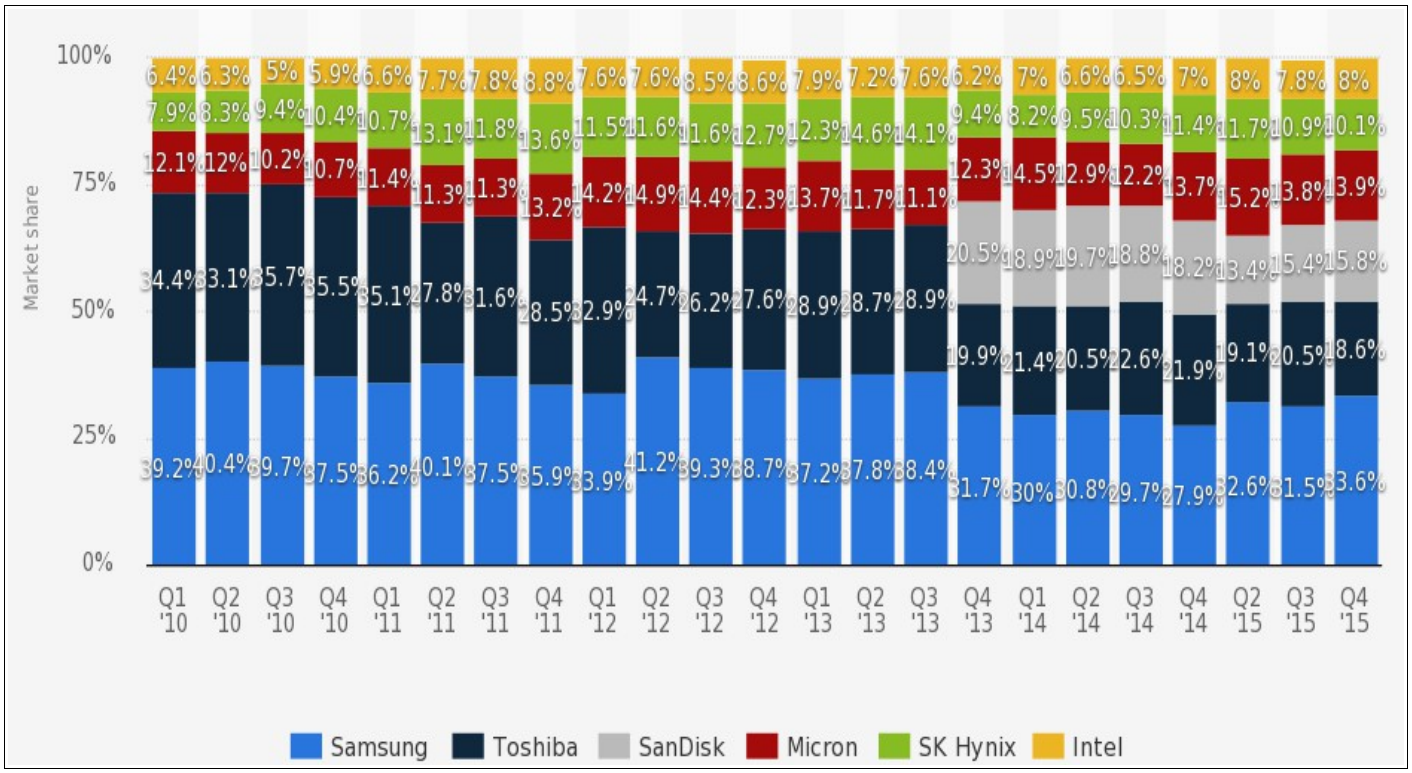


Fig. 12: *Cuota de mercado mundial por fabricantes de memoria NAND Flash (2010 - 2015)*

A partir de esta gráfica se decide centrar el anális en la oferta NAND de los 4 primeras marcas de la lista.

2.2.2.1.- Spansion/Cypress

En la gama de productos SLC de alto rendimiento y seguridad se incluyen dos familias estándar que soportan 1bit y 4 bit ECC. Disponibles en densidades de un 1Gb y 16Gb. Dispositivos orientados a la

industria Automotiva, instrumentación, electrónica de consumo y equipos de comunicación.

Ofrece otra familia SL de productos SecureNANDTM, destinada a aplicaciones más sensibles a los errores como es la ejecución de código de arranque, aplicaciones y almacenamiento de firmware, tipo SL.

Características típicas de las familias MS y ML de la gama SLC:

- Densities: 1Gb to 16Gb
- Voltages: 3 V and 1.8 V options
- Endurance: 100,000 P/E cycles (typical)
- Data Retention: 10 years (typical)
- ECC Requirements: 1-bit or 4-bit options
- Bus Width: x8 or x16 options
- Packages:
- 48-pin TSOP 12 mm x 20 mm
- 63-ball BGA 9 mm x 11 mm
- 67-ball BGA 8 mm x 6.5 mm
- Open NAND Flash Interface (ONFI) 1.0 compliant

Ofrece varias familias:

ML	25ns, 3.3V, 100,000 ciclos de vida con 1-bit (4Xnm) ECC (error correction code) o 4-Bit ECC (3X nm). SLC NAND soluciones seguras y eficiente para almacenamiento embebdido. Tamaño de página : (2048 + 64 spare) bytes. 1Gb a 8Gb y 1Gb a 16Gb
MS	45ns, 1.8V, 100,000 ciclos de vida con 1-bit (4Xnm) ECC (error correction code). SLC NAND soluciones seguras y eficiente para almacenamiento embebdido. Tamaño de página: (2048 + 64 spare) bytes. De 1Gb a 4Gb y de 1 a 16Gb
SL	4, 2 y 1Gb. 3.3V, 25 ns. Sólo con contactos de bola. No producción todavía.

Tabla 10: *Familia NAND cypress*

2.2.2.2.- Toshiba

Toshiba ofrece productos de memoria flash NAND SLC con una amplia gama de capacidades y tipos de encapsulado. Tecnología de alto rendimiento y resistencia de los grabados. Amplio rango de aplicaciones.

Ofrece dos líneas de nand flash:

- con controlador integrado eMMc, etc.
- SLC
  - 24 nm, el nodo de la tecnología más avanzada para SLC NAND: capacidades de 1, 2, 4 y 8 Gb disponible sólo en cantidades de producción.
  - Amplia gama de capacidades.
- BENAND
  - Incorpora la lógica ECC
  - 24nm
  - NAND con interfaz de serie.
- Con SPI
  - Compatible con el modo 0 y el modo 3 de la interfaz periférica de serie (SPI).
  - Incorpora la lógica ECC.

- Elimina la necesidad de un procesador host que realice la ECC y se puede configurar para proporcionar la información de corrección de ECC necesaria para administrar la memoria.
- El proceso SLC NAND de 24 nm más reciente.
- Amplia gama de capacidades.
- Disponible en cantidades de producción con capacidades de 1, 2 y 4 Gb
- Disponible en pequeños encapsulados: WSON de 6 x 8 mm, SOP de 10,3 x 7,5 mm, BGA(*) de 6 x 8 mm

Sin ECC (se hace desde el host)	BENAND, con ECC incorporado	Con SPI (sólo 4 bits y tsop 16)
512MB: TC58DVM92A5TAx0	1Gbit: TC58BVG0S3HTA00	2GB: TC58CYG2S0HQAIE
1GBit:TC58NVG0S3HTAx0	2gbit: TC58BVG1S3HTAx0	
2GBit:TC58NVG1S3HTAx0	4gbit :TC58BVG2S0HTAx0	
4GBit TC58NVG2S0HTAx0		

Tabla 11: *Tipos de NAND Toshiba*

2.2.2.3.- Hynix

Está lanzando la 1ª generación 3D NAND del 2014. En 2015, la segunda generación 3D NAND. Emplea tecnología de vanguardia en 3D NAND con la que pretende abrir nuevos espectros y posibilidades a todas las aplicaciones de almacenamiento. Cada año presenta una nueva generación de productos y se especializa cada vez más en este tipo de soluciones: 3D, almacenamiento masivo compacto SSD y eMMC. Para acceder a información de su catálogo y datasheets es preciso ponerse en contacto con la entidad.

2.2.2.4.- Samsung

Lo mismo ocurre con Samsung la única información sobre sus soluciones nand Raw se centran en una tecnología llamada V-NAND.
Samsung ha revolucionado la industria de almacenamiento NAND desplazando la estructura plana a una estructura vertical. Samsung tecnología V-NAND cuenta con un diseño único que apila 48 capas una encima de otra en lugar de tratar de reducir el tamaño de paso de las células. Samsung utiliza Channel Hole Technology para permitir a las células conectar verticalmente uno con el otro a través de un canal cilíndrico que se extiende a través de células apiladas.

La tecnología Samsung V-NAND supera las limitaciones de capacidad de la tecnología tradicional 2D NAND con su diseño vertical revolucionario. En V-NAND también se aplica la tecnología innovadora de Charge Trap Flash que previene la corrupción de datos causada por la interferencia de célula a célula.

La sinergia de ambas innovaciones estructurales y materiales conduce a una mayor velocidad, eficiencia energética y resistencia.

2.2.2.5.- Micron

Si una aplicación necesita velocidad y la resistencia por encima de todo, SLC NAND es la mejor opción. Alto rendimiento, aplicaciones de alta resistencia, cumple con las demandas de misión crítica. Ofrece una amplia cartera de productos con distintas características, funcionalidades y el rendimientos:

Gama	Descripción
SLC	Hasta 100.000 P / E de resistencia de ciclo, más rápido rendimiento que otras tecnologías NAND MLC y TLC, compatibles con la interfaz síncrona ONFi. Densidades desde 128mb hasta 512Gb.
MLC	Sólido rendimiento y la resistencia es dos veces la densidad de NAND SLC a un menor costo por bit. Compatibilidad con la interfaz síncrona ONFi.
TLC	Una mayor densidad en el mismo espacio, a un costo más bajo que SLC o MLC NAND
SERIE	Permite a las grandes subsistemas de memoria y aumentar la eficiencia de energía de alto rendimiento, bajo consumo de energía y mayor ancho de banda.
3D	Capacidades más altas. Mayor ancho de banda de lectura / escritura y la E / S, velocidades de consumo de energía reducido en modo de espera

Tabla 12: *Gama de NAND Micron*

Memorias SLC que usan típicamente en el foro de Texas Instrument y en las placas basadas en AM3358 de Texas y CircuitCo (Beaglebone):

8G	MT29F8G08ABABAWP	8b y 16b, 1,8V y 3,3V, 1b y 4b ECC
4G	MT29F4G08ABAEAWP	
2G	MT29F2G08ABAEAWP	

Tabla 13: *Ejemplos de modelos SLC*

Ofrece rápidas velocidades de lectura y escritura capacidades, excelente resistencia y algoritmos relativamente simples ECC.

2.2.3.- Elección del modelo

Uno de los factores determinantes es la apuesta por Micron que Texas Instruments y circuitCo (fabricante de beaglebone) y la confianza que depositan en este fabricante para integrar su modelo en sus placas de desarrollo basadas en AM335xx. Este hecho nos aporta unas ciertas garantías de compatibilidad.

A la vez, existen profesionales de la marca dando soporte en el espacio virtual de consulta que texas instruments ofrece a los usuarios registrados de forma gratuita. Por lo tanto, es más fácil hallar información y resolver cuestiones relacionadas con este modelo de memoria junto con las dudas sobre el bus GPMC.

Micron cuenta con un canal de distribución internacional importante, una gran flota de distribuidores oficiales (AVNET, Arrow, DijiKey, etc..) reconocidos disponen de este modelo en stock sin venderlo al pormayor y con unos plazos de entrega de 3 días.

Otro factor importante es la inteligibilidad e información detallada gráficamente y descriptivamente en su dasheet. Las operaciones, organización y funcionamieno del dispositivo están exaustivamente descritos y argumentados.

Se efectua el pedido del modelo **MT29F4G16ABADAWP**, cuya estructura y funcionamiento se describe a continuación.

Todas las operaciones NAND son iniciadas disponiendo un ciclo de comando, ciclo de direcciones y lectura o escritura de datos, tantos bytes como el comando enviado especifique. Existe un circuito de control de I/O que recibe estos datos.

Su datasheet ofrece una descripción gráfica completa y unas descripciones detalladas de cómo debe ser una transacción de todos sus comandos y dirección en nº de ciclos necesarios del modelo en cuestión:

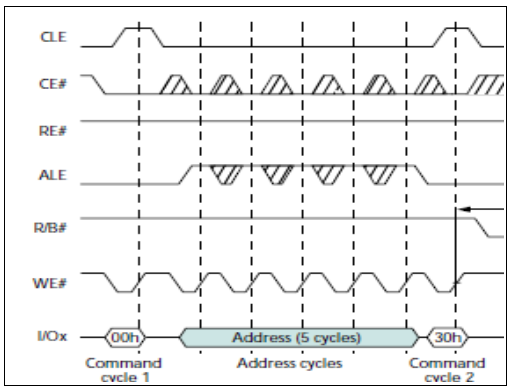


Fig. 13: *Petición de operación NAND*

La memoria seleccionada se basa en un dispositivo asíncrono SLC de 4GB (512Mbytes). La unidad mínima a la que un sólo chip enable puede acceder es una unidad lógica equivalente a un nand flash "die" (LUN). Por cada CE tenemos un LUN, por lo tanto nuestro chip es un LUN. Sus características y parámetros pueden variar respecto de dispositivos de mayores densidades. El arreglo/matriz de esta memoria se agrupa en 2 planos de series de bloques (que son la unidad más pequeña borrable dentro de una NAND). Borrar un bloque deja todos sus bits a '1' (FF..). Para cambiar el valor de '1' a '0' se precisa la acción de programación. La más pequeña entidad que puede ser programada de una vez es un byte.

La interfaz de datos multiplexada ofrece un pinout consistente para todos los modelos actuales y densidades. Este pinout permite a los diseñadores usardensidades inferiores y migrar a mayores densidades sin necesidad de cambios en el hardware donde se tiene integrado la nand. Se tienen otras dos señales (WP y RB) que controlan el hardware de protección de escritura y monitoreo del estado de dispositivo. Toda esta interfaz hardware crea un dispositivo con un pinout estándar que se mantienen independientemente de la densidad. Habilitando posibles migraciones a superiores densidades sin la necesidad de rediseños en la placa.

2.2.3.1.- Arquitectura y funcionamiento

Si se atiende al siguiente esquema sobre la arquitectura del dispositivo, los comandos, son pasados al registro de comandos. Luego, el comando pasa al circuito de control lógico que activa las señales apropiadas para controlar las operaciones de forma interna. Las direcciones son pasadas al registro de direcciones y enviado al decodificador de columnas o de filas. Sin embargo, los datos pasan hacia o desde la memoria (palabra a palabra) a través de dos registros: datos y cache. Datos y caché formarán un único registro cuando se den operaciones de página. Pero serán registros independientes con las operaciones caché. También se cuenta con un registro de status que informa de los estados de las operaciones del dispositivo. Estos registros y circuitos de control no forman parte del array en sí. Sin embargo los decodificadores y los registros de datos y caché, sí.

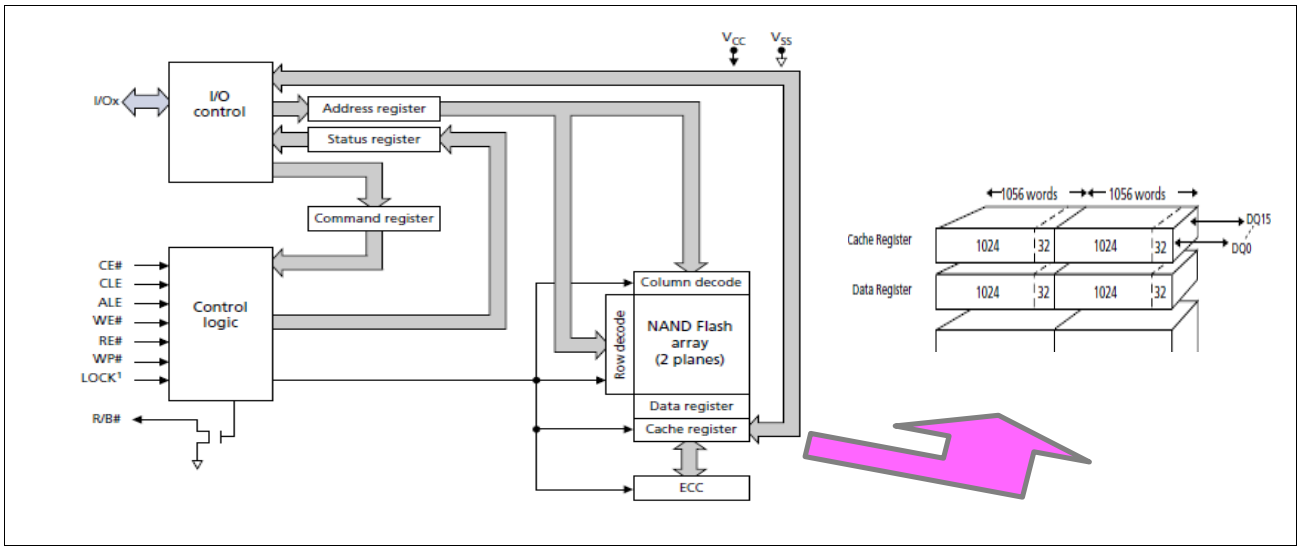


Fig. 14: *Arquitectura del dispositivo empleado*

En la operación de escritura el dato a ser programado es registrado en el registro de datos en el flanco de subida de la señal WE#. En la operación de lectura el dato es extraído del registro de datos de forma parecida pero con la señal RE#, responsable de extraer la actual dato e incrementar hacia siguiente posición. Comandos especiales son usados para acceso random o mover datos para hacer el acceso aleatorio posible durante la escritura. Los tiempos de WE# y RE# pueden alcanzar una velocidad de transferencia de 25 ns (por cada una). Cuando RE# o (CE#) no están activadas (nivel bajo), los buffers de salida están en tri-estado.

Una memoria NAND se programa y se lee por páginas. Pero se borra por bloque. En este modelo un bloque tiene 64KWord (2x64KB). En un plano hay 2048 bloques (2 planos en la memoria entera). Los 4GBit están organizados en bloques de 2048 con 64 pags/bloque. Cada página además contiene 2112 bytes, que en realidad consiste en un área de 2048-byte para datos y 64-bytes de "spare area". En el caso de bus x16=1024 palabras de 16b (1024 bytes x 2) y 32 x16b (64Bytes). Es un area típicamente empleada para ECC, wear-leveling, y otras funciones de sobrecarga del software. Aunque físicamente es el mismo que el resto de la página. Se tiene una interfaz de 16-bit. Los datos del lado host se conectan a la memoria por un bus de datos bidireccional de 16bits. Para dispositivo de 16b los comandos y direcciones usarán los 8 bits más bajos y los 8 más altos sólo se usan cuando tenemos (1 ciclo o varios) transferencias de datos.

Borrar bloques requiere una media de 700 us. Después de cargar el dato en el registro y programar una página requiere 200 us. Una operación de lectura requiere aprox 25 us, intervalo en el que la página es accedida desde el array y cargada en su registro de 2112 bytes (16896b). El registro está entonces disponible para el usuario para registrar el dato. La interfaz integra otras 6 señales de control. El dato es desplazado desde o hacia el registro de datos de 16 en 16 b de la NAND a la vez.

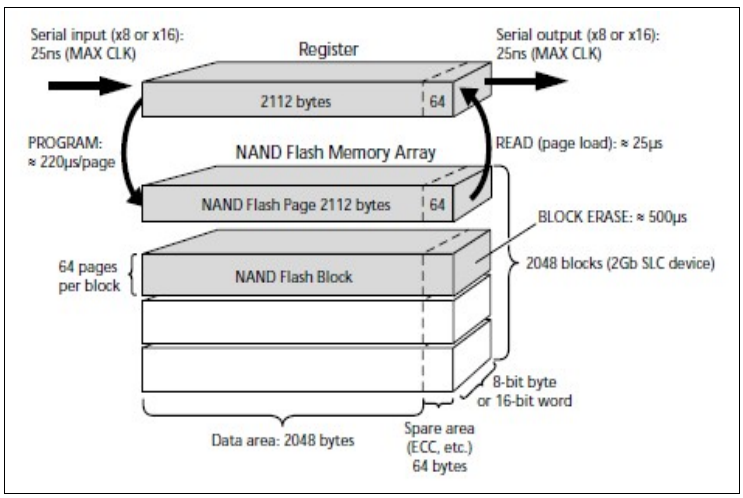


Fig. 15: *Transferencia de datos en lectura/escritura de una página.*

2.2.3.2.- Bus en alta impedancia:

La utilidad del tercer estado (Hi-Z) es suprimir la influencia de un dispositivo del resto del circuito. Si más de un dispositivo está conectado, poner una salida en Hi-Z se usa para que en un mismo bus no haya dos señales diferentes, es decir, una con valor 1 y otra con valor 0. Porque si ambas señales circularan por la misma línea, no podríamos determinar el valor que está circulando en la misma. Los buffer triestado también se usan para implementa multiplezores nte aquellos con un gran número de entradas. Las funciones del buffer triestado suelen ser útiles, principalmente para el diseño de componentes electrónicos con una cierta funcionalidad controlada internamente, como puede ser la implementación de un bus multiplexado, el cual puede usar buffer triestado para dejar activa solo una de sus entradas y anular las otras de forma que no interfieran en la salida. También se pueden usar los buffer triestado para dotar a los dispositivos de funcionalidad controlada externamente, es decir el usuario puede controlar esos buffer triestado para controlar el dispositivo. Este tipo de dispositivos tienen posibilidad de habilitación o deshabilitación de lectura, escritura o incluso del propio dispositivo. Cuando la salida del triestado están en estado de alta impedancia, su influencia en el resto del circuito es eliminada; pero también su valor binario se vueve desconocido. Si ningún otro elemento del circuito manda una señal hacia ese nodo, el bus permanecerá en alta impedencia con valor desconocido. Para evitar esto, los diseñadores de circuitos usualmente utilizarán resistencias pull up o pull-down (normalmente en el rango de 1 a 100K Ohmios) para determinar el valor en la salida del triestado, cuando éste está en Hi-Z, es decir, estas resistencias asignan el valor 1 ó 0 a la salida del triestado cuando está en alta impedancia. El driver GPMC permite tener en cuenta estas resistencias habilitando las resistencias internas de pullup y pulldown de cada uno de los pines del bus.

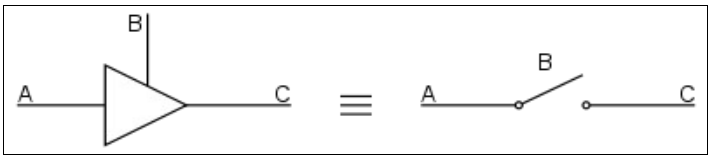


Fig. 16: *Buffer triestado*

Se puede ver un buffer triestado como un interruptor. Es decir; cuando en B hay un 1, funciona como si el interruptor estuviera activado, mientras que si hay un 0, actúa como si estuviera desactivado y el nodo A queda desconectado de C.

Cuando una de las dos señales no está activada, CE# and RE# se activan los buffers de salida, habilitando a NAND Flash para compartir el bus de datos con otros tipos de memoria. Hay memorias que necesitan CE activado todo el ciclo y otros que no, se pueden compartir con accesos a otras memorias cuando se da la situación de chip don't care (otros dispositivos conectados a este bus pueden ser accedidos mientras la memoria nand esté ocupada con sus operaciones internas). Esta propiedad es aprovechable y útil en diseños que necesiten múltiples dispositivos en el mismo bus. Cosa que en este diseño inicial no tendrá lugar. Pero abre la posibilidad a ampliar el sistema en un futuro ampliando y mejorando las funcionalidades del driver desarrollado en este trabajo.

3.- Metodología y desarrollo

Se ha anunciado en anteriores apartados que el esquema de interfaz aplicado es RAW NAND, que implica un desarrollo software más laborioso; pero también más personalizado. Si se tiene presente el esquema de la figura se observa que el diseño del software controlador puede implementar, además del driver propiamente dicho, los métodos de control de bloques fallidos y balanceo de uso de bloques.

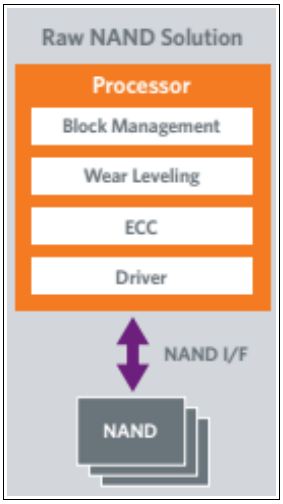


Fig.17: *Interfaz Raw*

A pesar de ser todos métodos implementables, no son bloques obligatorios para realizar transacciones entre el periférico y el uP/uC, aunque sí altamente recomendables. De hecho, en soluciones avanzadas de otros autores son opciones que pueden habilitarse y deshabilitarse; pero se desaconseja omitir su uso. El presente trabajo pretende desarrollar un driver básico, con las mínimas prestaciones necesarias para acceder a la memoria mediante una aplicación sencilla y poco exigente si se compara con el verdadero campo de aplicaciones para el que las prestaciones flash son una necesidad. La aplicación de prueba no forma parte de un sistema crítico, ni exigente en términos de volumen de demandas de operaciones ni de comandos complejos. Una lectura o escritura fallida por accesos a bloques defectuosos no controlados, todavía no supondría una consecuencia grave en ninguna parte del sistema ni del entorno en el que se integra. Se ha desarrollado únicamente el código del bloque DRIVER, es decir: una versión limitada; pero ampliable. El software queda abierto a mejoras y puede ampliarse para trabajar con requisitos más exigentes.

Tal como se ha adelantado anteriormente, el presente trabajo requiere desarrollo de software y hardware. La planificación del proyecto ha seguido una secuencia de fases cronológicamente dispuestas, ya que de seguir otro orden no habría sido realizable. Aunque algunas de ellas pueden llevarse a cabo de forma concurrente, como es el caso de hardware y software ya que son bloques independientes y pueden testearse por separado en fases previas a su integración.

3.1.- preparación del entorno de desarrollo

a) Familiarización con entorno original a través de la realización de algunas aplicaciones sencillas a modo de tutorial de entrenamiento para entender el funcionamiento de los elementos y deducir una sistemática de trabajo lo más eficiente posible:

- Beaglebone White
- Acceso remoto por terminal Lnux (Linux shell: Ubuntu y Anstrong)
- CAPE-BONE-TT01V1
- Software: compilador y archivos makefile

b) Migración de OS Angstrom a Ubuntu embedded. Se recuerda que el siguiente material es brindado:

- **BeagleBoneUbuntu4GB_V05.rar**: la imagen de la BB.
- **Win32DiskImager-0.9.5-binary:** programa para instalar desde Windows la imagen que se puede obtener gratuitamente desde la siguiente ubicación https://sourceforge.net/projects/win32diskimager/

Secuencia de pasos para instalar la nueva versión Ubntu embedded en la BeagleBone:

1. Descomprimir **BeagleBoneUbuntu4GB_V05.ra**r
2. Descomprimir **Win32DiskImager**
3. Hacerse con una microSD de al menos 4GB e insertarla en el PC
4. Abrir la aplicación Win32DiskImager.
5. Seleccionar la imagen **BeagleBoneUbuntu4GB_V05.iso** como archivo de origen en Win32DiskImager
6. Seleccionar la microSD como disco destino en Win32DiskImager
7. Pulsar el botón "write".

Al cabo de unos minutos queda restaurada la imagen original. Para finalizar se extrae la microSD del PC y se inserta en la BB. Se arranca el portátil y se activa el arranque con Ubuntu. Al conectar la BB al portátil esta arrancará desde la flash con el nuevo sistema instalado y lista para ser accedida desde un terminal (siempre que se conecte a través del cable de red y la comunicación se establezca previamente). Una de las cosas que cabe tener en cuenta es que la IP asignada es 192.168.2.100 y que la configuración de network de Ubuntu ya que puede dar lugar a problemas de conexión entre los dos hosts. Existen varias formas de proceder se edita mediante la terminal en modo root el archivo *interfaces*.

- etc > network > interfaces

```
gedit interfaces
```

- asignar los siguientes valores de la imagen y guardar:

interfaces ×

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
#activar primera linea tb si no funciona
auto eth0
#iface eth0 inet dhcp
iface eth0 inet static
address 192.168.2.1
netmask 255.255.255.0
network 192.168.2.0
broadcast 192.168.2.255
```

Fig. 18: *Archivo configuración ethernet en host portátil*

- en la terminal lanzar las siguientes órdenes para restaurar los valores:

```
ifconfig eth0 down
ifconfig eth0 up
```

También es necesaria una clonación en Ubuntu la versión de kernel usada para compilar con el nuevo compilador *arm-linux-gnueabihf-gcc* que permite generar los archivos binarios para la nueva plataforma de la BB. La versión anterior compilaba para kernel de Angstrom 3.2.42 Ubuntu en el equipo portátil para instalar y emplear el compilador cruzado. La instalación del SDK completo con el fin de construir aplicaciones y el sistema BEAGLEBONE no será necesaria. Sólo la cadena de herramientas (tool chain) compilador cruzado es suficiente con el fin de hacerlo. El compilador GCC utilizado para este trabajo

es proporcionado por Linaro. Linaro es organización de ingeniería sin fines de lucro que trabaja con el código abierto y el software libre. Su actividad se especializa en el núcleo de Linux y GNU Compiler Collection, entre otros.

Creación de los nuevos makefiles:

Los archivos en los que se basa cualquier proyecto de este SDK constan del código fuente de la app y los includes de librerías y cabeceras que sea necesario incluir. Cuando alguna de las aplicaciones requiere que el sistema reconozca un driver, como en el presente proyecto ocurre, se debe añadir el código fuente del driver a esa lista de archivos. Los makefiles son archivos que permiten al compilador generar los archivos binarios ejecutables a partir de sus códigos fuente. Por lo tanto un makefile debe estar destinado a compilar la app, otro para compilar el módulo con su carpeta de inlcudes. Opcionalmente, y por comodidad, puede crearse otro en el directorio principal para ejecutar los otros dos. Estos 3 makefiles se mostrarán en el apartado de anexos.

d) Operando con el entorno en modo red local. Una vez, se dispone de los ejecutables llega el momento de transferirlos a la BB para ser gestionados y ejectudos por su sistema. La pauta que se ha definido es:

1. conectarse a la BB a través del comando SSH en una nueva terminal con permisos root:

```
ssh ubuntu@192.168.2.100
```

   - logearse con la PW: ubuntu
   - crear una carpeta en el directorio raíz p ej: carpetaName (~/carpetaName1)

```
cd ~/

mkdir carpetaName1

mkdir carpetaName2
```

2. en otro terminal ir al directorio del workspace donde se encuentra los códigos fuente en PC
   - compilar: se generan los archivos .ko (si hay que incluir un driver) y el ejecutable de la app
   - pasarlos a la BB a través del comando scp:

```
scp nombreArchivoEjecutable ubuntu@192.168.2.100: ~/carpetaName1

scp nombreModulo.ko ubuntu@192.168.2.100:~/carpetaName2
```

3. Si el proyecto incluye driver, una vez en la BB de nuevo acceder a ~/carpetaName2 con permisos root (*sudo -s* y pw: ubuntu) cargar el driver:

```
insmod nombreModulo.ko
```

   - comprobar que se ha cargado mediante comando: lsmod
   - para suprimir el driver aplicar el comando: rmmod nombreModulo.ko: realizar este paso antes de cargar una nueva versión del driver SIEMPRE.

4. desde la carpeta que contiene el ejecutable de la app ejecutar la app:

```
./nombreArchivoEjecutable
```

e) Test: generación de ficheros y ejecución de aplicación de prueba.

Tras cargar el módulo module.ko y ejecutar la aplicación app del material de prácticas, efectivamente se obtiene el resultado esperado. Queda validada la migración hacia el nuevo sistema y el entorno de desarrollo diseñado en el trabajo en apartados anteriores mencionado.

3.2.- diseño de circuito impreso PCB

3.2.1.- Elección de software CAD de diseño PCB.

Se requiere de un entorno de desarrollo que permita generar los archivos necesarios para iniciar su fabricación. Pasa por instalar un software de diseño que facilite la edición de piezas SMD y pistas con precisión, que procese la información y lo traduzca en archivos compatibles con la maquinaria de fabricación. Los formatos de archivo más extendidos en la industria de la producción de PCBs son los conocidos como GERBER y sus variantes.
La elección se disputa entre dos marcas propietarias orCAD e Eagle light Edition 7.5.0.

Puntos fuertes del orCAD:	**Puntos fuertes Eagle light Edition 7.5.0**
Software de los más empleados y conocidos en el ámbito docente y comercial	También es un conocido en la industria
Dispone de una versión gratuita	Tiene una versión gratuita con limitaciones
La unidad de la universidad que interpreta y traduce los diseños trabaja con esta herramienta de diseño PCB	Hallar documentación oficial y guías rápidas intuitivas (castellano e inglés) no es complicado
	Farnell o Dijy Key, proporcionan paquetes para Eagle con los símbolos de gran variedad de dispositivos y componentes de su catálogo
	Gran cantidad de componentes en la librería
	Empleado de forma más reciente por el desarrollador

Tabla 14: *OrCad Vs. Eagle light Ed.*

La suma de factores puede ahorrar una cantidad importante del costo temporal. La familiarización con el modo de empleo de este programa no supondría un punto de partida excesivamente novedoso como puede ser con el caso de orCAD para el desarrollador.
La versión gratuita brinda las siguientes posibilidades de diseño:

1. Área máxima: 10x8 cm
2. Nº de capas máximo: 2 (top y bottom)
3. No puede emplearse para usos comerciales ni fines lucrativos.

3.2.2.-Familiarización con el *soft cad*

Familiarización siguiendo tutoriales muy intuitivos y detallados, con ejemplos sencillos que permiten reconocer las herramientas necesarias del software. Se ha trabajado con la creación de food prints de componentes y asociarlos a sus símbolos esquemáticos, añadir valores y nombres, propiedades a los pines (in, out, both), escoger la escala del grid (importante), traducir el diseño de esquemático a PCB, etc. Inicialmente se realizó el equema; pero se observa que resulta más fácil y menos confuso iniciar el diseño directamente en modo PCB. Las conexiones se establecen automáticamente mediante unos hilos de color amarillo - llamados *signal*- cuando se aplica la traducción, partiendo de un circuito esquemático.

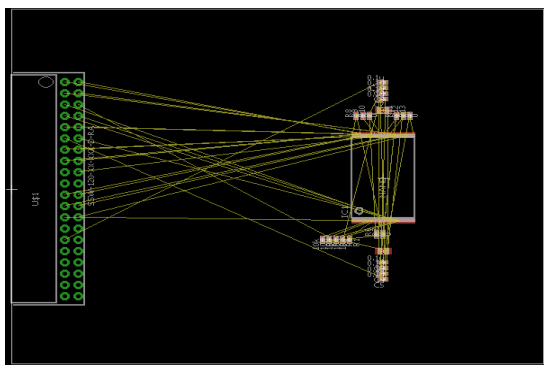


Fig. 19: *Trazado automático de signals*

Según la orientación y ubicación del foodprint del chip de memoria puede implicar un ruteo más o menos complicado. Este hecho ha llevado a iniciar diferentes propuestas hasta hallar la disposición por la que el trazado sea más óptimo y menos problemático (menor nº de cruces de pistas y vías). Lo que ha llevado a determinar que era positivo conectar los componentes mediante las signals (hilos amarillos) de forma manual en el modo PCB, en lugar de hacerlo de forma previa mediante un esquemático (modo esquema).

Este software, como la mayoría, dispone de un autorute y detección de errores. Pueden automatizarse las restricciones de fabricación para obtener los avisos pertinentes. El presente diseño tiene sus propias restricciones teóricas y las de fabricación. No obstante, la experiencia recomienda que teniendo una cantidad de espacio suficiente con un nº de elementos relativamente mínimo el ruteo y seguimiento de restricciones se realice de forma manual porque es más efectivo.

c) definición de conexionado entre pinaje de la beaglebone y dispositivo. Se revisa el conexionado entre las señales de los headers P9 y P8 de la BB y los conectores de la CAPE-BONE-TT01V1 para detectar posibles incoherencias.

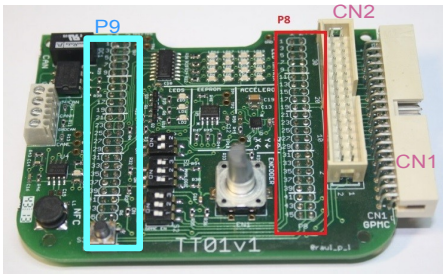


Fig. 20: *Cabeceras y contactos*

Se comprueba la disponibilidad de todas las señales que el acceso NAND y NOR necesitan (en conector CN1). CN2 es un conector CN1 que facilita medidas con un analizador lógico. CN1 es un conector macho por el que se conecta el bus GPMC de la BB con una placa de desarrollo Altera DE2 basada en FPGA. Teóricamente las señales GPMC en P8 y P9 son llevadas a los conectores CN1 y CN2 por el trazado de pistas de esta Cape. Pero se debe corroborar la presencia de todas las señales y su correspondencia con el pinaje del modelo NAND para tomar medidas oportunas antes de iniciar el diseño. La siguiente lista de correspondencias se han obtenido en este punto.

C1	C2	SG Cape	BB SG	P9	P8	Observaciones
38		GND	GND			
37		SPI1_CS1	SPI1_CS1	42		
36		SPI1_cs0/SPI	SPI1_s_CLK	28		Dos señales diferentes
35		GND	GND			
34		SPI1Mosi	SPI1Mosi	30		
33		GND	GND			
32	4	BE1	BE1	12		
30	2	GND	GND	NC	NC	
29		VCC3.3V	SPI0_SCLK	22		Incoherencia: 2 señales diferentes
28	6	CSN2	CSN2		20	
27	3	WP	W0	11		Incoherencia: en realidad es W0
26	8	CSN0	CSN0		26	
25	5	AD8	AD8		19	
24	10	AD1	AD1		24	
23	7	AD0	AD0		25	
22	14	AD5	AD5		22	
21	9	AD4	AD4		23	
20	16	CLK	CLK		18	
19	13	CS1N	CS1N		21	
18		GND	GND	NC	NC	
17	15	AD11	AD11		17	
16	17	AD15	AD15		15	
15	18	AD14	AD14		16	
14	19	AD9	AD9		13	
13	20	AD10	AD10		14	
12		GND	GND	NC	NC	
11		VCC5.5		NC	NC	No existe conexión con P8/P9
10	21	AD13	AD13		11	
9	22	AD12	AD12		12	
8	25	BE0N/CLE	BE0_CLE		9	
7	26	WEN	WEN		10	
6	27	ALE	ALE		7	
5	28	OEN	OEN		8	
4	29	AD2	AD2		5	
3	30	AD3	AD3		6	
2	31	AD6	AD6		3	
1	32	AD7	AD7		4	
11	__	WP	WP	13		No salida en Cape. Solución: conector con hilo
39	__	VCC3.3V	VCC3.3V	3,4		No salida en Cape. Solución: conector con hilo
40	34	GND	GND	1,2	1,2	Conectado a plano de masa Cape.

Tabla 15: *Seguimiento de señales entre BB (P8, P9) y la Cape-bone-TT01v1*

El tramo de casillas violeta de la tabla (de la 33 a la 38) no son empleadas en la interfaz NAND.

11 y 39 son señales que no se proporcionan en el conector CN1, 27 se indica como señal WP cuando en realidad proviene del pin wait0. CN1 tiene algunos pines sin conectar y se usarán para llevar las señales de las dos señales 11 y 39 mediante un puenteo provisional:

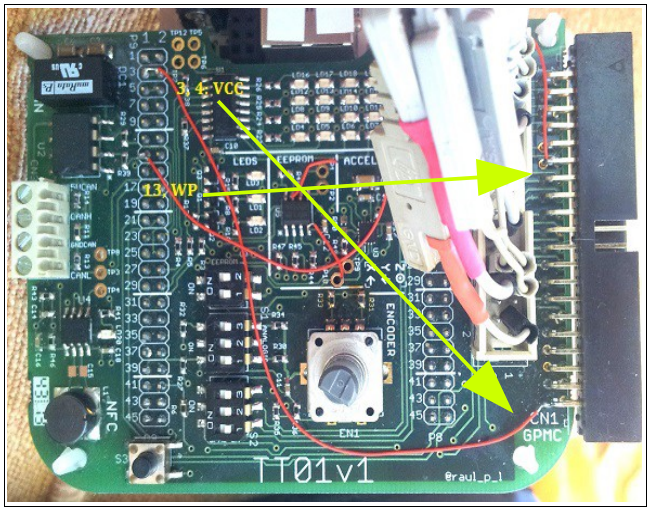


Fig.21: *Puenteo WP y VCC en la Cape-bone-TT01v1*

Conexionado entre el conector CN1 (al otro lado de las líneas discontinuas de la imagen) de la Cape-bone-TT01v1 y el pinaje del dispositivo NAND empleado. Las señales en violeta no son fuente de preocupación puesto que, así, ya se puede formar el bus GPMC para NAND.

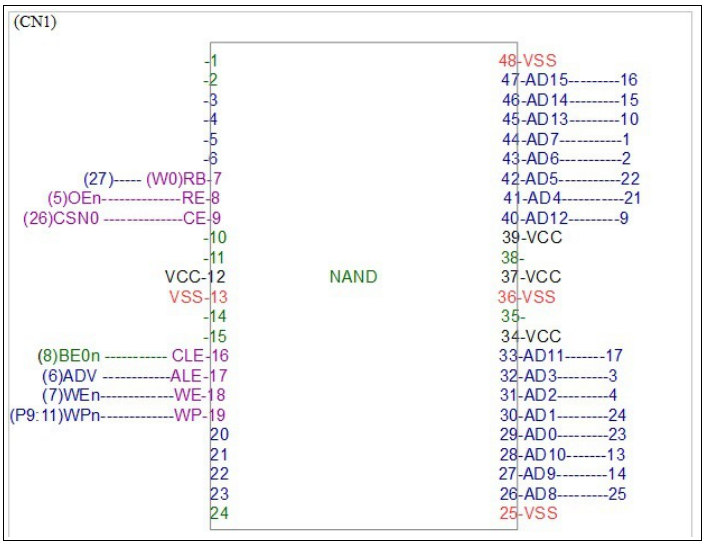


Fig. 22: *Correspondencia de pines CN1 - NAND*

Cabe decir que el pinaje del conector CN1 se replican en el conector hembra de la placa PCB fabricada. En lugar de CS2, se han empleado CS1 y CS0 para habilitar el dispositivo de memoria. El pin CLK no se emplea al tratarse de una memoria asíncrona. Sin embargo, este chip necesita una señal WP y una alimentación de 3.3V que no se encuentra replicada en el conector porque la placa DE2 tiene su propia alimentación. Debido a ello se realiza el puente comentado anteriormente desde P9.

3.2.3.- Restricciones y criterios de fabricación:

1. Distancia entre pistas no puede superar los 0.2 mm, es la máxima resolución de la fresadora
2. Los agujeros no están metalizados, deben rellenarse de estaño: no pueden ir componentes sobre las vías
3. Sólo se puede usar cara top y cara bottom
4. No pueden usarse planos de masa ni de alimentación
5. 4 líneas de alimentación: se necesitan 4 desacoplos para filtrar picos de corriente e interferencias procedentes de los armónicos irradiados por las conmutaciones presentes en la placa.

6. Resistencias pull up para señales de control negadas

7. Condensadores cerámicos conectados directamente a los pines de alimentación

8. Formar filtros con Rs (>10 Ohm) con el pin de alimentación y condensadores.

9. R jumper es una Rs de valor reducido, ayuda a favorecer el filtrado con los condensadores en pines VCC sensibles al ruido. Ayuda a mejorar (atenuar) el pico de resonancia (factor Q) provocado por la inductancia parásita que pueda producir la impedancia de una capacidad a partir de una determinada frecuencia (ZC tiende incrementar su inductancia). Aunque en este caso, tal vez no sea necesario; no hará mal prevenir.

    - Fresonancia = 1/2pi(ESL·C)^1/2

    - Q (factor de calidad de un filtro) mide el pico de resonancia y puede medirse = (2·pi·L)/R

        - Una forma de reducir dicha Q es incluir una Rs en la línea alimentación lo más cerca posible y lo más pequeña posible.

10. Condensadores de desacoplo lo más cerca posible de los pines de alimentación: los más grandes primero y los menores en parelelo para mejorar ancho de banda. Sobretodo con pistas equidistants. Típicamente: uno de 10uF y una batería de 0,1uF; cuanta más cantidad de condensadores mejor el ancho de banda será.

11. Crear áreas de baja impedancia, como polígonos de masa y alimentación, o reducir el trazado de las pistas por las que conectan los condensadores de desacoplo para reducir la inductancia. Tratar de no realizar pistas en forma de espiral.

12. Para evitar el cruce de pistas, las pistas de una cara de la PCB seguirán una horientación y las de la otra cara la orientación perpendicular.

3.2.3.1.- Deducción de Rpull up

Son usadas normalmente en electrónica digital para asegurarnos que los niveles lógicos que tenemos a las entradas de los dispositivos son correctos. Si no conectáramos las entradas del circuito en Pull-Up, estas entradas quedarían en un nivel lógico indeterminado, sobretodo cuando una de las señales procede de un pin en drenador abierto. El propio fabricante ofrece explícitas instrucciones de insertar una Rpull up para el correcto funcionamiento.

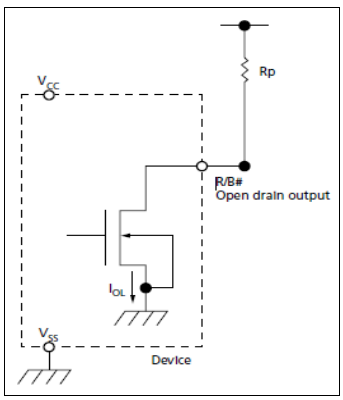


Fig. 23: *Circuito en drenador abierto de la señal RB.*

Cuando RB transiciona a estado LOW (ocupado), el driver de la señal habilita la salida RB negada a ser influida por una OR. Ya que, R/B# suele estar conectada a un pin de interrupción del host controlador. La combinación de Rp y la capacidad de carga del circuito determinará su tiempo de subida (tc=CxR). El valor de Rp dependerá de los requisitos del sistema porque valores elevados de Rp pueden causar retardos considerables en la señal.

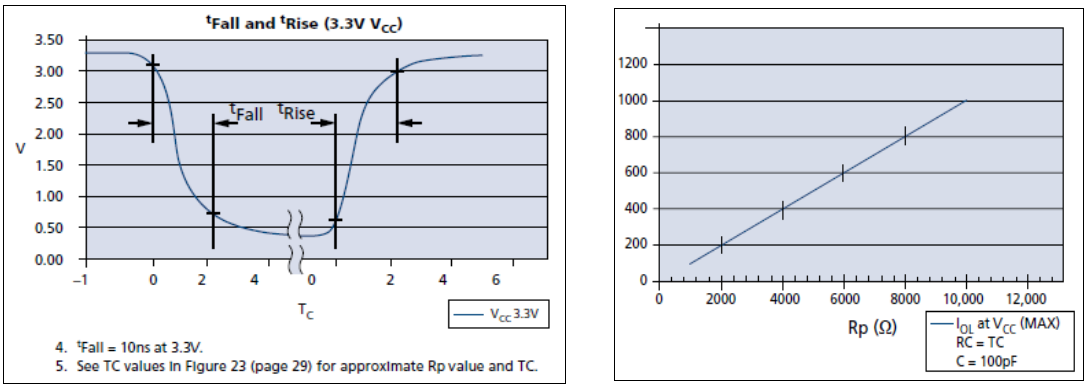


Fig. 24: *tiempo de caída y de subida de RB en tC y tC en función de Rpull up*

- Si tf= a 2 ·Tc = ~10ns, Tc =~ 5ns.
- Con Rp = (5KOhm :10KOhm) → Tc = (5ns :10ns)
    - Rp = 10kOhm es un valor típico.

3.2.3.2.-Filtros de acoplo.

- La capacidad equivalente de una batería de 4 C de 0,1uF + 1C de 10uF = 25nF.
- La frecuencia de corte de cada filtro de desacoplo: R·C = 1/(2·pi·C·fc) → fc=12Mhz
    - Suprime los armónicos producidos por ondas cuadradas de 40Mhz**3.2.4.- lista de componentes para adaptación de dispositivo:**

REF DESIGNATIORS	PART	PCB FOOTPRINT	DESCRIPTION	MAN	PART #
C	10uF,10V	805	CAP CER 10UF 10V Y5V 0805	TDK	C2012Y5V1A106Z
R	10K,1%	402	Resistor 10Kohm 1/16W 5% 0402	Rohm	MCR01MZPJ103
RJ	0R	402	Resistor Zero ohm Jumper 0402	Yageo	RC0402JR-070RL
C	0.1uf,6.3V	402	Capacitor 0.1uF 16V 10% 0402 X7R	Kemet	C0402C104K4RACT
Conector	Post Socket S	2.54 mm	Receptor, 40, RA	Samtec	SSW-120-02-G-D-RA

Tabla 16: *lista de componentes definitiva*

3.2.5.- diseño final PCB; cara top, cara bottom:

- top: rojo
- bottom: azul

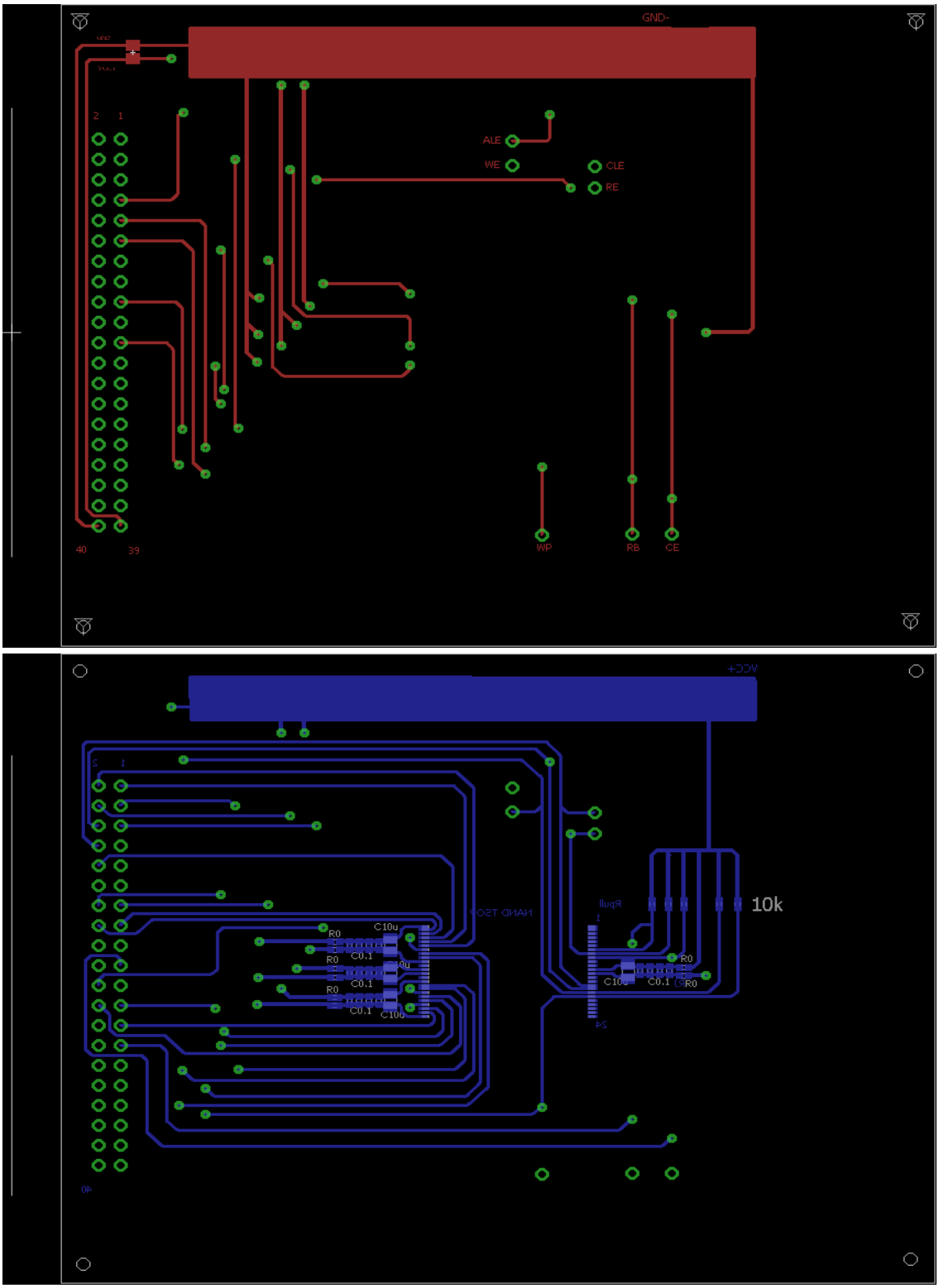


Fig.25: *caras top y bottom del diseño final*

3.2.6.- Generación de ficheros GERBER para la fabricación:

Una vez se tiene el diseño final se procede a la generación de archivos GERBER para iniciar su fabricación. El programa permite generar varios formatos para alcanzar la mayor compatibilidad con los diferentes modelos de fresadora. En concreto, el staff de fabricación del centro docente sugiere que se seleccione el formato apto para GERBER_RS274X. De otro modo el fabricante podría no visualizar correctamente el diseño y por lo tanto sería incapaz de reproducirlo.

En la carpeta del proyecto del diseño activa se accede a la herramienta CAM y se abrirá una ventana:

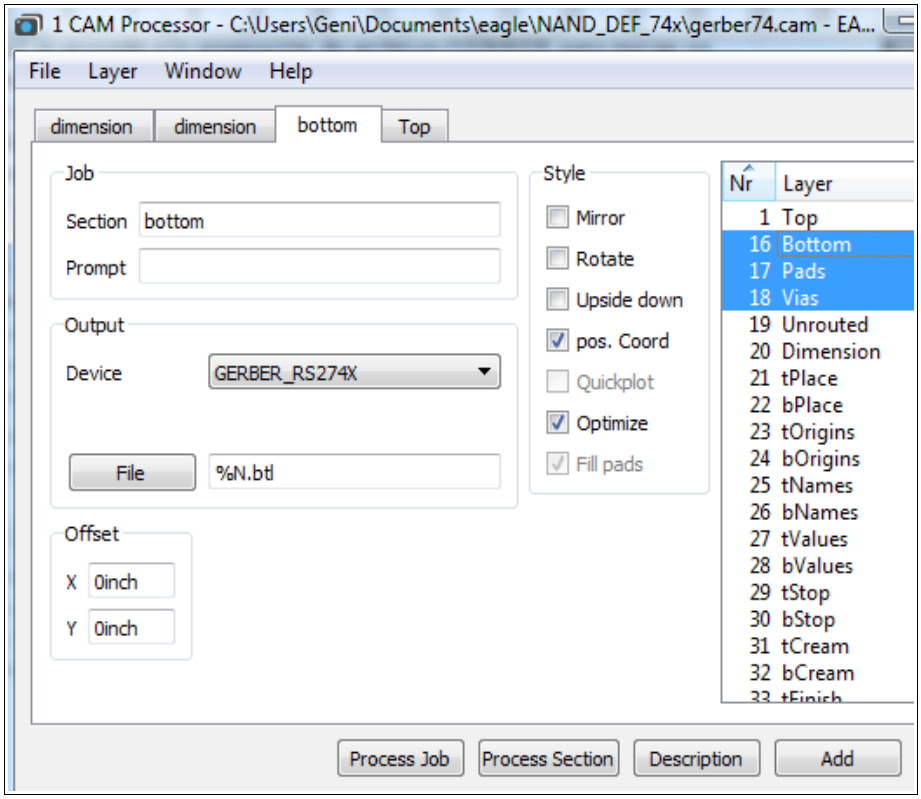


Fig. 26: *generacion de archivo de la capa bottom*

Se repetirá este procedimiento para generar los archivos de la capa Top, Bottom y de las dimensiones. Lo cual se indicará en cada caso en el campo "Section" y se asociará con la capa en cuestión del menú desplegable "layer". Ahí se escogerán las capas que se deseen incluir en dicha sección, como información para el fabricante. 3 elementos de información son suficientes: capa, Pads y Vías. En la sección de dimensiones bastará con incuir la capa 20 "Dimension". %N.btl es el archivo resultante con su extensión correspondiente.

Cada sección se irá añadiendo mediante ADD para que consten cuando se procesen estos datos. Tras la capa de dimensiones se pueden generar sus archivos: "Process Job". Al cerrar esta ventana se puede guardar el archivo CAM en el directorio del proyecto para reutilizarlo o prescindir de hacerlo. En todo caso no es necesario el archivo para la fabricación, tan sólo los archivos resultantes de procesar el conjunto de secciones.

El siguiente dato, imprescindible para el fabricante, es la ubucación y tamaños de todos los taladros que irán en la placa. El proceso difiere en algunos puntos del primero y por ello es preciso abrir un nuevo CAM: cam>file>new>job. Algunos tutoriales recomiendan escoger el fichero CAM ("cam processor job file"): excellon.cam. Este paso llevará a abrir una ventana similar a la imagen anterior. El device destino debe ser "excellon rack". Este solicita un fichero origen, porque dicha información la obtiene de un archivo cuya extensión es ".drl". De otro modo no puede interpretar la información origen. Por lo tanto, es el propio usuario quién debe ofrecer este archivo. Una forma de obtenerlo es mediante el intérprete de comandos y scripts que Eagle nos brinda. La línea de comandos se encuentra en la parte superior de la ventana del esquemático o de la placa, tal como se puede apreciar en la primera de las imágenes, escribiendo: *run drillcfg*. Se recomienda escoger la unidad de medida

que el fabricante utiliza. En el caso actual, se guardarán las medidas en pulgadas. Por comodidad y seguridad se guardan todos los archivos generados en la carpeta del proyecto.

Desde la ventana del trabajo CAM se asocia el archivo .drl anterior y se selecciona la capa de “drills” y “hole” en el menú “layer”. Se procesa el trabajo y así se obtiene el último de los ficheros necesarios. En la segunda imagen, se puede observar toda la lista de archivos generados dentro de la carpeta del proyecto, la cual será entregada al fabricante, íntegramente.

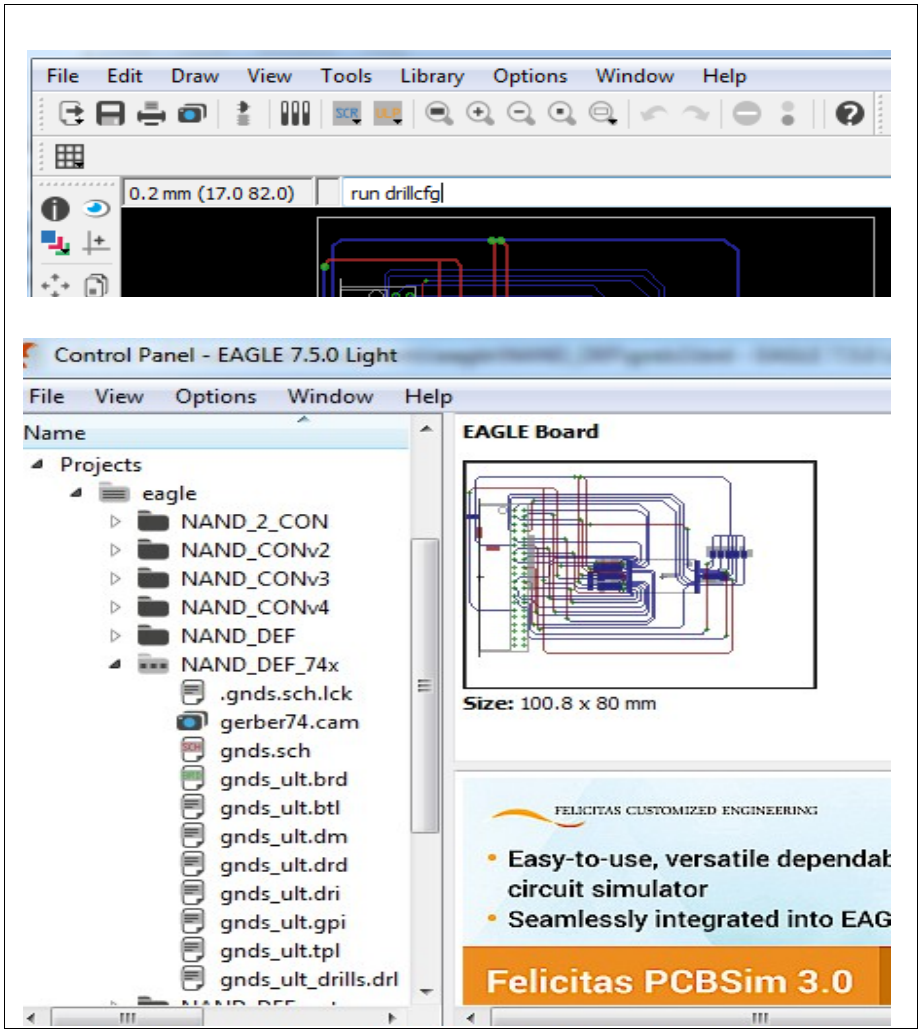


Fig. 27: *a) Línea de comando para generar taladros,*
b) directorio de archivos de fabricación generados

3.2.7.- Comprobación continuidad de las pistas trazadas:

Las dimensiones de los componentes del tipo de encapsulado SMD ha requerido precisión microscópica (empleo de microscopio). El proceso de fabricación y soldadura SMD se ha desarrollado con éxito y la continuidad ha sido verificada mediante téster en modo diodo contactando con un polo en los pines de origen (cabeceras P8 y P9 de la CAPE) y destino (contactos de la placa desarrollada). En este punto la placa está lista para conectarse en la interfaz de la CAPE, hacer pruebas de transferencias de datos desde la Beaglebone y ajustar los componentes de adaptación de señales si fuese necesario. Observar **Anexo1** con imágenes del resultado de fabricación.

3.3.- Programación del driver GPMC

Este bloque es uno de los puntos fuertes y núcleo del desarrollo del conjunto del sistema. Constituye el software que controla la interfaz. Este reto se puede dividir en dos aristas, en controlador y elemento controlado, origen y destino o GPMC host y lado del periférico NAND.

Se citan a continuación los rasgos más relevantes y requisitos del lado GPMC, driver GPMC. Ha sido preciso un estudio exaustivo de la configuración del GPMC de la beaglebone para acceso NAND a partir de TRM del AM335xx Sitara para comprender el funcionamiento de estos estándares antes de proponer un plan de diseño de código fuente.

Un dispositivo NAND requiere múltiples etapas de programación de direcciones y uno o más comandos. La programación de las etapas de comandos y fases de dirección del driver GPMC se realizan mediante operaciones de escritura a las ubicaciones de los registros GPMC_NAND_COMMAND_i y GPMC_NAND_ADDRESS_i con los valores correctos de comando y dirección en cada ciclo. Estas posiciones de memoria están mapeadas en la región del CS asociada con el registro. El control de temporización del chip select deben ser programadas de acuerdo con las especificaciones del fabricante del chip.

La unidad MPU, es responsable de lanzar el nº de comandos y direcciones mediante una correcta programación del software driver, acorde con el set y el esquema de direccionamiento del modelo escogido. Por lo tanto, se procede a tomar nota de las indicaciones y especificaciones del fabricante para deducir los valores y parámetros relacionados con el timing de las señales de control con los que programar el driver.

3.3.1.- Memory Side: Chip de memoria NAND MT29F4G16ABADAWP

Estudio del protocolo de acceso flash NAND específico del modelo de memoria empleado a partir de su datasheet. En este apartado se muestra cómo son las secuencias de operaciones que requiere cada tipo de acceso y los timings de las señales de control.

Comandos, direcciones y datos son registrados en la nand en el flanco alto de reloj de WE#. HIGH CLE indica que un ciclo de comandos está teniendo lugar. HIGH ALE significa que un ciclo de ADDRESS INPUT está teniendo lugar. Toda operación se inicia con un ciclo de comandos. Una secuencia de comando consiste en un ciclo de latch del comando, varios ciclos de address input y uno o más ciclos de datos (en modo escritura o lectura). El ciclo de comando asíncrono se realiza llevando el comando a la parte baja del bus I/O (7:0). El comando se escribe desde el bus hacia el registro de comandos durante el flanco ascendente de WE#, con CE# a nivel bajo y CLE a nivel alto (ALE bajo y RE# alto). Mientras que los ciclos de escritura de dirección se escribe desde el bus I/O[7:0] al registro de dirección en flanco ascendente de WE# cuando CE# es bajo, ALE alto (CLE bajo y RE# alto).

El dato se escribe desde el bus I/O[15:0] hasta el registro cache de la LUN (unidad de memoria gestionada por un CS) en el flanco ascendente de WE# cuando CE# es bajo (ALE bajo, CLE bajo y RE# alto). El dato puede ser extraído desde el registro cache de una LUN hacia el bus I/O en el flanco descendiente de RE# cuando CE# es bajo, ALE es bajo, CLE es bajo y WE# alto. El dato puede ser leído (extraído) desde el LUN si la memoria se encuentra en estado READY.

3.3.1.1.- Consideraciones:

a) WP

Si WP#=0 → Borrar y programar = deshabilitadas.

Si WP#=1 → Borrar y programar = habilitadas.

Durante encendido se recomienda que WP = 0 hasta que VCC sea estable (unos 150us) → previene programaciones y borrados involuntarios.

El estado de WP sólo puede transicionar:

- Cuando el dispositivo no está no está ocupado RDY (o RB#)= 1
    1. Antes del inicio de una secuencia de comandos.
    2. Después de una secuencia de comando completada

Después de experimentar un cambio en WP# el host tiene que dejar tWW antes de lanzar un nuevo comando.

b) R/B - RDY

Se puede determinar el status de la memoria en un momento concreto solicitando el estado consultando el registro de status (campo/bit RDY) o monitoreando desde el host (GPMC side) la señal hardware directamente (R/B).

- R/B# = 1 → RDY=1 → dispositivo "listo"
- R/B# = 1 → 0 → RDY=0 → dispositivo está "ocupado".

c) Comportamiento de WR/RD – ADD/COMMAND:

- Las solicitudes de salida de datos suelen ser ignorados por una matriz (LUN) si está ocupada (RDY = 0): los datos pueden emitirse desde un (LUN) si se encuentra en un estado READY.
- Los datos se escriben en el registro de datos en el flanco ascendente del dispositivo si no está ocupado. La entrada de datos se ignora si CS=1 o RDY = 0.
- Las direcciones y los comandos serán ignorados por el (LUN) si está ocupados (RDY = 0).

d) Direccionamiento:

Muchos comandos requieren un nº de ciclos de dirección seguidos de un ciclo de comando. RESET y READ STATUS son la excepción ya que requieren menor nº de ciclos además de que puede ser solicitados y ejecutados mientras el dispositivo esté ocupado, cosa que el resto no. Nuevos comandos no deben ser reclamados hasta que la memoria deje de estar ocupada en las transacciones pendientes. El direccionamiento sigue una serie de fases: El primer y segundo ciclos (bytes) especifican la dirección de columna, que especifica el primer byte de la página. Si la última columna es 2112, la dirección de esta última posición será 08h en el segundo byte y 3F en el primer byte. PA[5:0] especifica la dirección de página dentro del bloque y las BA [16:6] la dirección de bloque. La dirección completa de 5 bytes se lanzan para la mayoría de operaciones de escritura y lectura, sólo dos ciclos (2bytes) son necesarios para operaciones especiales que acceden al dato de forma aleatoria dentro de la página.

3.3.1.2.- Sincronismo de control de la memoria según su fabricante:

Dos son los tipos de accesos que conforman una operación: programación de comandos y dirección y la fase de datos (escritura y lectura).

1.- latch de comandos y un latch de direcciones:

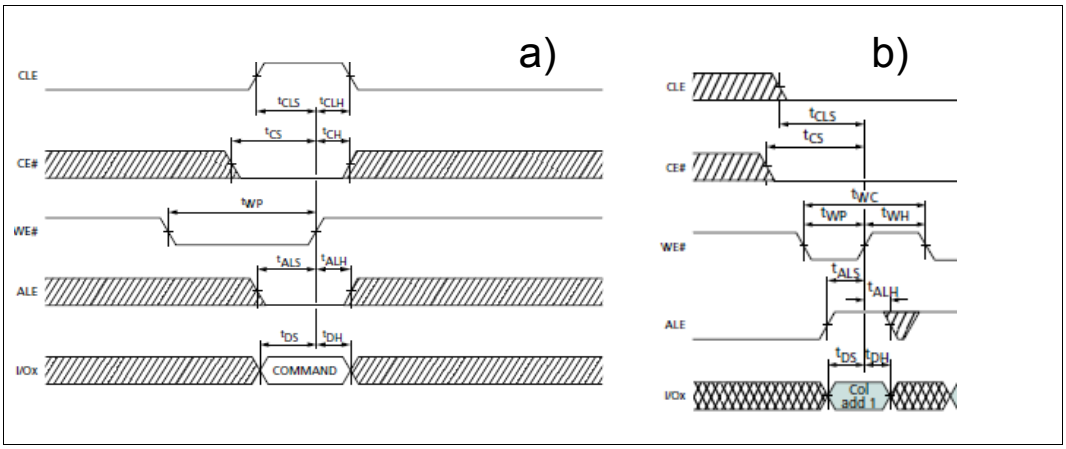


Fig. 28: *a) ciclo de command latch, b) ciclo de address latch*

Durante un latch de comandos y de direcciones todos los timings de las señales de control participantes toman como referencia el flanco ascendente de la señal WE.

Relación de tiempos mínimos (ns):

tCLS	10	Tiempo setup CLE para su activación. En desactivación tiempo hasta flanco ascendente WE
tCLH	**5**	el tiempo de hold de comando que ha de pasar para desactivarse (o activarse) después de tWP.
tCS	15	tiempo setup hasta WE desde la activación de CE
tCH	**5**	Tiempo hold (desde WP high) para la desactivación de CE
tWC	20	es el ciclo completo de un ciclo de escritura (estado bajo + alto)
tWP	10	Pulso en estado bajo WE (de flanco descendiente a flanco ascendente): activa escritura de dato en bus
tWH	7	Pulso en estado alto WE: tiempo hasta la siguiente activación de WE# (siguiente ciclo)
tALS	10	Tiempo setup ALE antes de activación WE
tALH	5	tiempo de hold ALE: tiempo desde WE high (tiempo de desactivación después de validar dirección),

Tabla 17: *Tiempos de setup y de hold para comandos*

2.-Input/output (WR/RD) de datos:

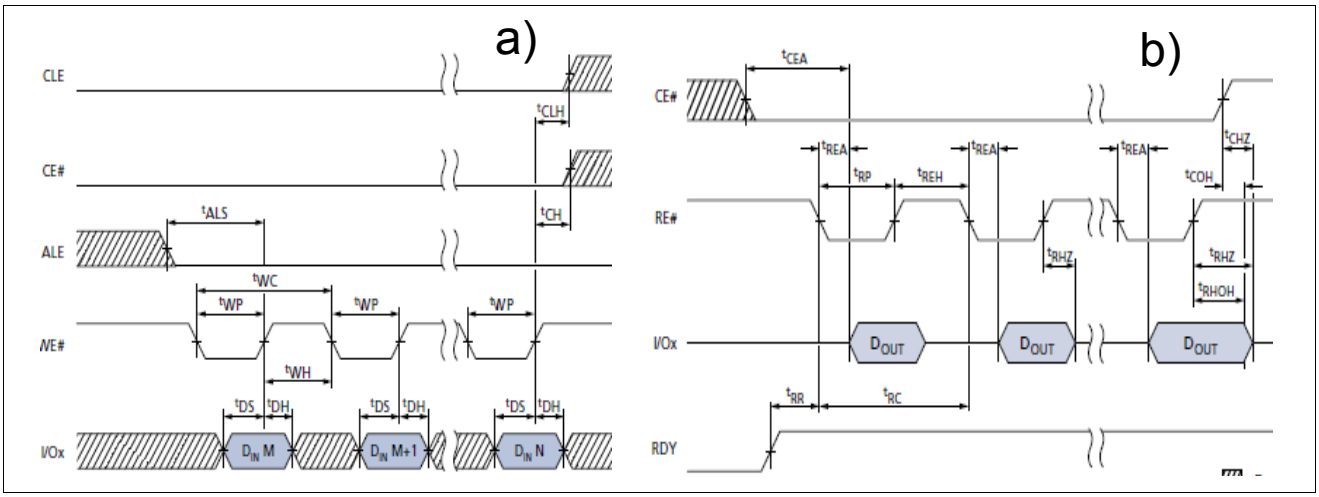


Fig. 29: *a) ciclos de escritura, b) ciclos de lectura*

Cuando la salida de dato se deshabilita o **CE se desactiva** → I/O flota en **alta impedancia**. Deben tenerse en cuenta los tiempos terminados en HZ.

Relación de tiempos mínimos (ns):

tALS	10	Tiempo desde la desactivación de ALE (ciclo direcciones) hasta 1er flanc asc WE por dato en bus
tRC	20	Tiempo que dura un ciclo RE# de lectura de un dato (entre dos flancos de bajada seguidos)
tRR	20	Tiempo entre flanco subida RB y flanco de bajada de RE#
tRP	10	Tiempo de pulso RE# bajo
tREH	7	Tiempo entre flanco ascendente y descendiente de RE#
tREA	16	Tiempo de acceso (que tarda el dato en estar presente en el bus) desde flanco desc RE#
tRHZ	100	Tiempo entre flanco ascendente RE# y bus en estado de alta Z (fin de tiempo de dato)
tRHOH	15	Tiempo de flanco ascendente RE# y tiempo de hold de modo output.
tCEA	25	Tiempo de acceso de CE# (que tarda en aparecer el primer dato) desde CE# descen.
tCOH	15	Tiempo de flanco alto CE# (desactivación de CE) hasta tiempo de hold del modo output.
tCHZ	50	Tiempo de flanco alto CE# hasta fin de último dato (inicio del bus en alta Z)

Tabla 18: *Tiempos para la escritura de dirección*

La lectura se activa por flanco descendiente de RE, aunque dependiendo de los tiempos escogidos puede darse en flanco ascendente o al siguiente pulso:

1. El controlador host (nuestra beagle) que usa tRC >= 30ns → el host puede registrar el dato en el flanco ascendente de RE#.
2. Si el controlador host usa un tRC < 30ns → el host puede registrar el dato en el siguiente flanco descendiente de RE#.

Se dispone de un set de comandos proporcionados por el fabricante con sus protocolos de petición. Algunos requieren un nº menor de ciclos de escritura. Por lo tanto se iniciarán las primeras pruebas con los comandos más sencillos e imprescindibles para la inicialización del sistema. El primer lugar se programará el driver GPMC del host. Acto seguido se comprobarán las formas de onda de las señales de control y los datos del bus por el analizador lógico. Finalmente, cuando las señales se comporten según lo esperado se conectará el host al dispositivo NAND de forma que comprobaremos si la interfaz opera correctamente.

3.- Operaciones de inicio:

Petición de RESET y petición de estado del chip

a) El comando RESET:

- pone la memoria en una condición conocida y aborta el comando en progreso.
- debe ser enviado a la memoria como el primer comando tras el encendido.
- hace que sea posible para abortar borrado y programación en proceso o bien vuelva a emitir el comando en un momento posterior.

Después del reset:

1. R/B# = 0 durante tRST (después de escribir al registro de comandos)
   - Primer reset tRST = 1ms
   - Posteriores tRST = 5us
2. READ, PROGRAM y ERASE pueden ser abortadas mientras el dispositivo está ocupado.

3. El registro de comandos queda limpio y preparado para el siguiente comando.
4. Los contenidos de los registros de datos y de caché quedan marcados como inválidos.
5. El registro de status contiene el valor:
   - Si WP# = HIGH → E0h = 1110 0000
   - Si WP# = LOW → 60h = 0110 0000

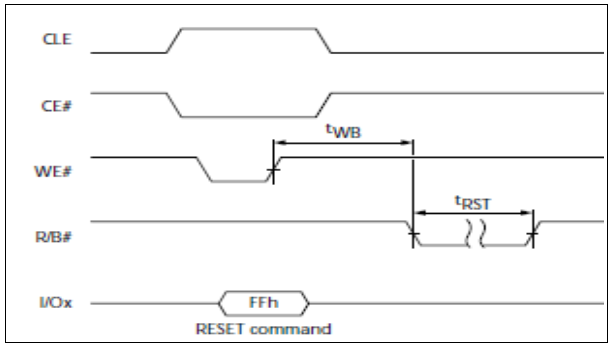


Fig. 30: *Lanzamiento comando RESET*

Relación de tiempos máximos en ns:

tWB	100	tiempo entre estado alto de WE a flanco alto de RB
tRST	5-10-500 us	Tiempo de reset para lectura-escritura-borrado. La duración del 1er reset puede llegar a 1ms (1000us)

Tabla 19: *Tiempos del comando de reset*

b) READ STATUS (70h) puede ser atendido mientras el dispositivo se encuentra en "busy". No requiere ni ciclos de dirección ni comando de cierre. Cuando sólo se quiere conocer el estado del bit RDY el comando debe ser lanzado una sola vez. Este estado puede ser releído:

- a) emitiendo la sincro de RE#
- b) manteniendo RE# en estado bajo sin conmutar a estado alto

El comando READ STATUS ofrece el estado de la señal write-protect signal y fallo/correcto en operaciones de PROGRAM/ERASE. READ STATUS (70h) retorna el estado del (LUN). Para acceder al registro de estado y leerlo, primero hay que habilitar la salida (output) del registro. El contenido del registro se escribirá en la salida I/O en cada petición de lectura de datos. Los cambios en el registro son vistos en el bus I/O(7:0) tanto tiempo como CE# = RE# = LOW. No es necesario conmutar RE# para ver una actualización del registro.

Consultar el registro de estado sirve para determinar cuando la transferencia de un dato desde la memoria al registro de datos ha finalizado (tR). Cuando la operación ha terminado, si no se ha deshabilitado el modo "status outuput", el bus permanece ocupado por el valor de este registro. Para que el host pueda capturar los datos después de una lectura monitoreada se tiene que deshabilitar el registro de estado, lanzando el comando READ MODE (00h).

Hay dos tipos de acceso a este registro, avanzado o básico. Dado que sólo se dispone de una LUN bastará con el modo básico: 70h.

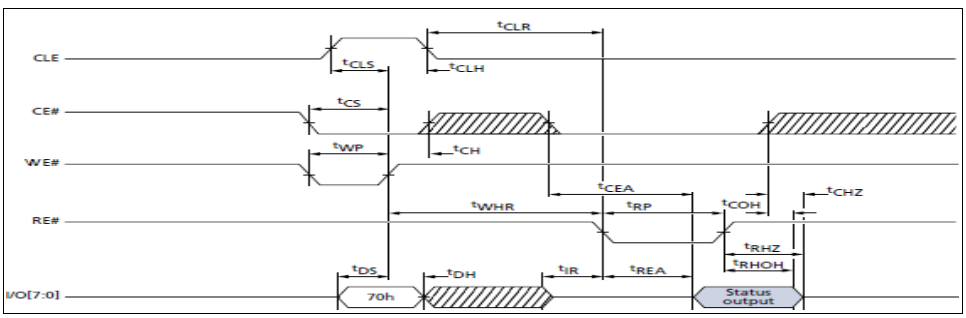


Fig. 31: *Operación comando read status*

Relación de tiempos mínimos (ns):

tWHR	60	Tiempo desde WE alto (latch comando) hasta RE bajo de la primera lectura después de la escritura comando (delay de lectura tras escritura)
tIR	0	Output High-Z to RE# LOW
tRP	10	Pulso (negado) total de RE
tCLR	10	Delay entre la desactivación de CLE y la próxima activación de RE
tCEA	25	Tiempo máximo de acceso desde activación CE a dato en bus IO
tREA	16	Tiempo máximo desde RE low hasta que aparece el dato en el bus.

Tabla 20: *Parámetros temporales para la lectura de estado*

Las descripciones del significado de cada bit del registro STATUS en 3 operaciones son:

St. **R**eg.	Programación de pag/Lect de pag/Borrado de bloque
7	0=WP, 1=no WP
6	0=Busy, 1=Ready
5	0=Busy, 1=Ready
3	(Sólo pag RD) 0=normal/incorregible, 1= "Re-Write" recomendable
1	0=Correcto, 1=Error
0	0=Correcto, 1=Error
Resto de bits	0/don't care

Tabla 21: *Significado de los campos del registro de estado de la memoria*

4.- Operaciones de Lectura:

Lectura de ID y de página

a) Comando read ID es un comando seguido de un sólo ciclo de dirección, que pueden ser 2:

- 00h: devuelve 5 bytes de identificador
- 20h: devuelve 4 según el estándar ONFI

Tiempos máximos (ns)

tAR	10	Delay entre último ALE y primer RE

El comando 00h indica la activación de la salida de datos y el inicio de una operación de lectura.

b) El comando de lectura de página es el comando 00h–30h. Copia una página de la matriz NAND a su registro caché y habilita la salida de datos. Los pasos para lanzar este comando y completar la lectura son:

1. escribir 00h en el registro de comandos
2. escribir los 5 ciclos de dirección en el registro de dirección
3. se produce una transferencia interna de los datos durante tR: memoria ocupada
4. Se puede determinar el progreso de la transferencia
    - consultando la señal RB
    - comando de estado: 70h → implica que cuando el pin RDY = 1, el host deshabilitará la salida del estado y habilitará la salida de datos cuando se lance el comando 00h.

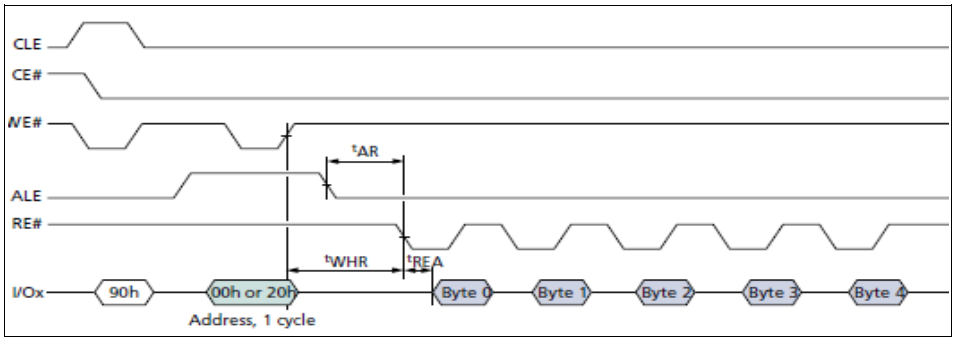


Fig. 32: *Sincronismo de lectura de ID*

5. Una vez el host inicia la captura del dato, la salida de datos se iniciará desde la columna de la página del bloque especificada en la dirección enviada, pasando los datos al bus. Este último paso sucede sólo si el host pide el dato mediante la lectura de su registro de datos, ya que dicha acción es la que activa el sincronismo de lectura.

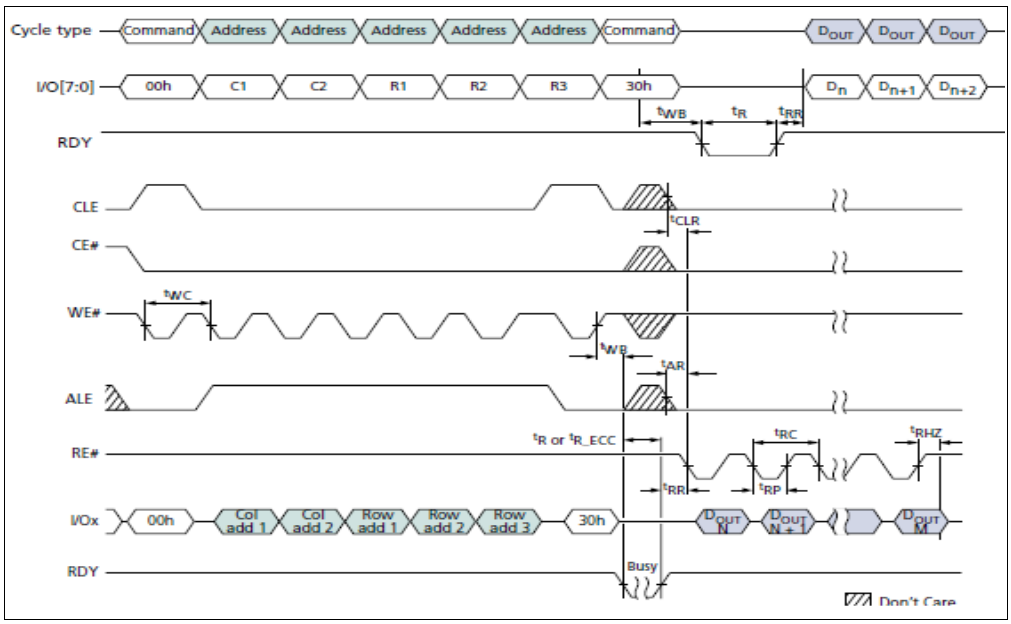


Fig.33 : *Sincronismo de activación de una lectura de página*

Tiempos mínimos en ns:

tRHW	100	Flanco ascendente RE (últ dato) a primer WE flanco descendiente (1r comando)
tR	25us	Transferencia de datos desde memoria a registro de datos (desde ultimo ciclo escritura WE = 0 → 1 hasta RB = 0→1)
tRR	20	Delay entre paso de busy a ready hasta primer flanco desc de RE#
tWB	100	Tiempo máximo entre último WE ascendente y a chip ocupado

Tabla 22: *Tiempos de una lecura de página*

5.- Operaciones de Escritura:

Borrado de bloque y escritura de página

a) El comando de borrado de bloque es 60h-D0h y se usa para limpiar el contenido de todo un bloque antes de escribir en cualquiera de sus páginas; borra el bloque entero.

Para borrar un bloque deben seguirse estos pasos:

1. Escribe 60h en el registro de comandos.
2. Escribe 3 ciclos de direcciones de fila (bloque): las columnas y páginas son ignoradas.

3. Escribir D0h en registro de comandos para cerrar operación.
4. Permanecerá en estado busy (RDY = 0) durante tBERS hasta que se haya borrado.
5. Revisar el progreso de la operación revisando el estado de RDY.
    - consultando RB
    - lanzando comando read Status
6. Cuando RDY = 1, revisar el estado del bit FAIL del registro STATUS.

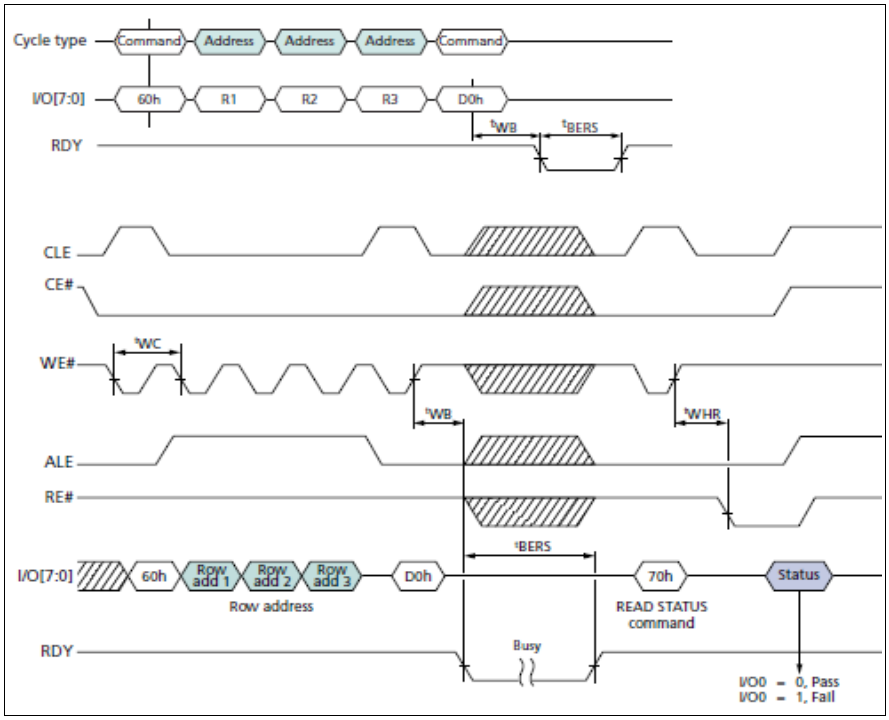


Fig. 34: *Sincronismo de borrado de bloque*

Tiempos máximo en ms:

tBERS	3	Tiempo de operación de borrado

Una operación de programación mueve los datos desde su registro de datos y/o caché hacia el array NAND. El contenido de los registros de datos y/o caché son modificados por el circuito de control lógico interno. Cuando se quiere programar una secuencia de páginas dentro de un mismo bloque el orden debe ser: desde el nº de página menos significativo hasta el mayor (0, 1, 2, ….., 63).

b) El comando de programar una página es: 80h-10h y programa una página desde el registro cache hacia una página del array NAND.

El comando habilita al host para introducir el dato en el registro cache y luego mueve el dato dsde el registro cache a la página del bloque especificada en la dirección. Así, los pasos de esta operación pueden secuenciarse así:

1. Escribir 80h en el registro de comandos: limpia el contenido del registro caché de la LUN
2. Escribir 5 ciclos de dirección.
3. Se suceden varios ciclos de input de datos en serie: se introducen a partir de la columna indicada en la dirección.
4. Después del último dato se debe escribir el comando 10h.
5. La LUN permanecerá ocupada durante tPROG hasta que el dato es transferido.

6. Cuando la LUN ya está en estado READY (RDY =1) el host debería comprobar el bit de FAIL del registro STATUS para verificar si la operación se ha completado con éxito.

   - consultando la señal RB
   - comando read STATUS

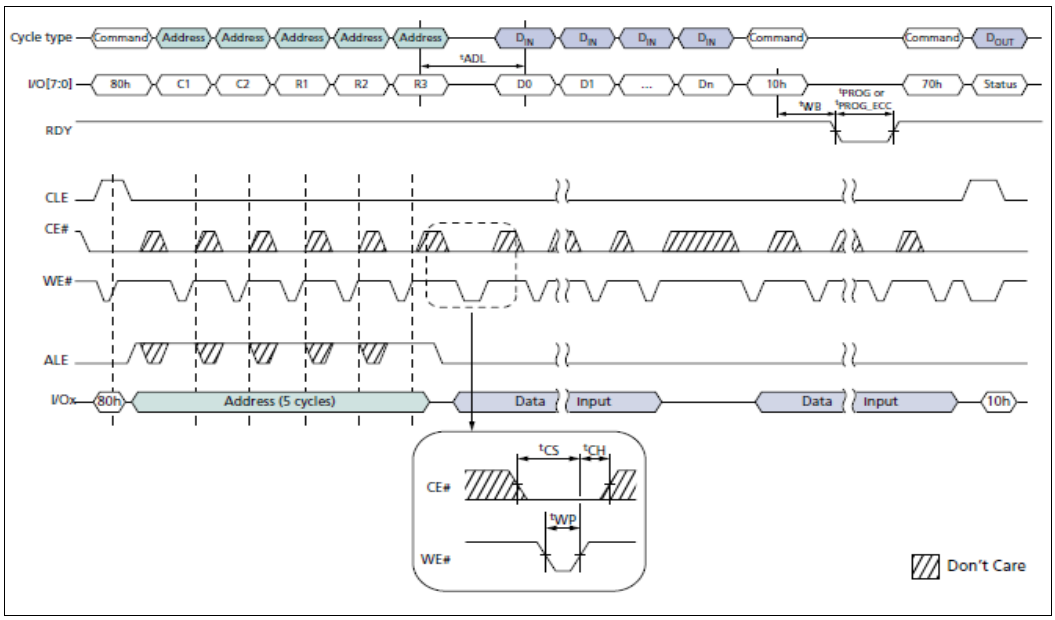


Fig. 35: *Sincronismo de una operación de programación de página*

Tiempos mínimos en ns:

tADL	70	(tiempo entre dirección y dato) en un ciclo de dirección tiempo entre la desactivación de WE (flanco subida) y el primer flanco de subida de WE para la entrada de datos en el bus, después de disponer y validar la dirección.
tPROG	600us	Max tiempo de programación de más del 50% de las pags

Tabla 23: *Tiempos de una escritura de página*

6.- Operaciones de inicialización:

Se recomienda proteger de escritura el dispositivo después del encendido mientras VCC no se haya estabilizado para evitar borrados/escrituras indeseados. Se indica que la alimentación es estable 100us después de alcanzar VCCmin en una rampa ascendente. WP está activo por defecto y el dispositivo se habrá alimentado al conectarlo a CN1 tiempo antes de iniciar el driver. Por seguridad se pausará el sistema mientras transcurren 200us antes de activar WP y lanzar el primer comando.

Para llevar a cabo la etapa de encedido se seguirán las siguientes fases:

1. Esperar 200us con Wp activado por defecto (protección de escritura/lectura)
2. Desactivar Wp desde registro de GPMC_CONFIG
3. Lanzar solicitud de comando de RESET
4. Esperar un milisegundo.
5. Lanzar solicitud de comando de lectura de estado READ STATUS.
6. Una vez libre lanzar comando de recuperación de ID.

Cuando se ha cargado el driver, el dispositivo ha arrancado y se ha inicializado, se puede empezar a escribir y leer una página. Para llevar a cabo una etapa de escritura y de lectura deben seguirse las fases que se han citado anteriormente.

Resumen:

Pasos	Escritura	Lectura
1	Lanzar comando de inicio	Lanzar comando de inicio
2	5 ciclos de dirección	5 ciclos de dirección
3	Escribir datos (16b-words **x** *n* posiciones de memoria de la pag) en DATA reg	Escribir comando de cierre
4	Escribir comando de cierre	Esperar tWB
5	Esperar tWB	Sondear señal RB hasta estar desactivada (estado alto)
6	Sondear señal RB hasta quedar desactivada (estado alto)	Leer datos (16b-words **x** *n* posiciones de memoria de la pag) de DATA reg
7	Lanzar comando READ STATUS para saber si se ha producido un error	

Tabla 24: *Resumen de las etapas de la operación de escritura y de lectura que el host debe implentar.*

3.3.2.- Host Driver Side

El driver GPMC en el lado del host se va a implementar para cargarse y gestionar el hardware GPMC de la CPU Sitara en el kernel de Linux Ubuntu embedded versión 3.8 porque es la última versión que incorpora el cape manager. Por lo tanto, es la última recomendada si se trabaja con una CAPE. En Linux, los drivers sofware son conocidos y tratados como módulos. La extensión del fichero de un módulo compilado en Linux es *.ko*.

Cada hardware es diferente de un fabricante a otro. Para que un sistema operativo se entienda con diferentes dispositivos existe el concepto de driver. Un driver soporta los programas que tienen contacto directo con el hardware del sistema del que forman parte. Esto implica, muchas veces, operaciones de escritura y lectura de los registros -de control y de datos- del controlador de un determinado hardware. Sus operaciones también cambian de un modelo a otro. La labor del driver es compatibilizar toda esa variedad y diferencias, cuando emplean la misma interfaz física. El driver que pretende montarse en el sistema es el controlador del bloque hardware GPMC de un AM335x, de arquitectura ARM y va a programarse siguiendo las especificaciones de su manual de referencia.

El GPMC puede gestionar hasta 7 regiones de memoria independientes mediante las señales CSi (0:6). Cada una de estas regiones tiene una serie de registros asociados a cada uno de forma independiente que permite configurar el protocolo de acceso asociado a dicho CS, programando sus parámetros. La codificación del driver gira entorno a la escritura de valores desde el código a estos registros. Los campos de configuración del protocolo empleado se hallan agrupados en diferentes registros, que son los siguientes con sus respectivas direcciones de base (los campos que los conforman con los valores posibles se hallan perfectamente detallados en el capítulo de registros del TRM de la CPU).

0h GPMC_REVISION	No modificado (valor por defecto)
10h GPMC_SYSCONFIG	
14h GPMC_SYSSTATUS	
18h GPMC_IRQSTATUS	No modificado (valor por defecto)
1Ch GPMC_IRQENABLE	
40h GPMC_TIMEOUT_CONTROL	
44h GPMC_ERR_ADDRESS	No modificado (valor por defecto)
48h GPMC_ERR_TYPE	No modificado (valor por defecto)
50h GPMC_CONFIG	
54h GPMC_STATUS	
60h GPMC_CONFIG1_i	Comunes en los 7 CS
64h GPMC_CONFIG2_i	
68h GPMC_CONFIG3_i	
6Ch GPMC_CONFIG4_i	
70h GPMC_CONFIG5_i	
74h GPMC_CONFIG6_i	
78h GPMC_CONFIG7_i	
7Ch GPMC_NAND_COMMAND_i	Sólo se usan en protocolo NAND.
80h GPMC_NAND_ADDRESS_i	
84h GPMC_NAND_DATA_i	

Tabla 25: *Lista de los registros de configuración del driver GPMC*

El manual de la CPU propone el siguiente flujo de implementación del software de control para el bus GPMC del uP, cuando se trata de acceder a memorias tipo flash.

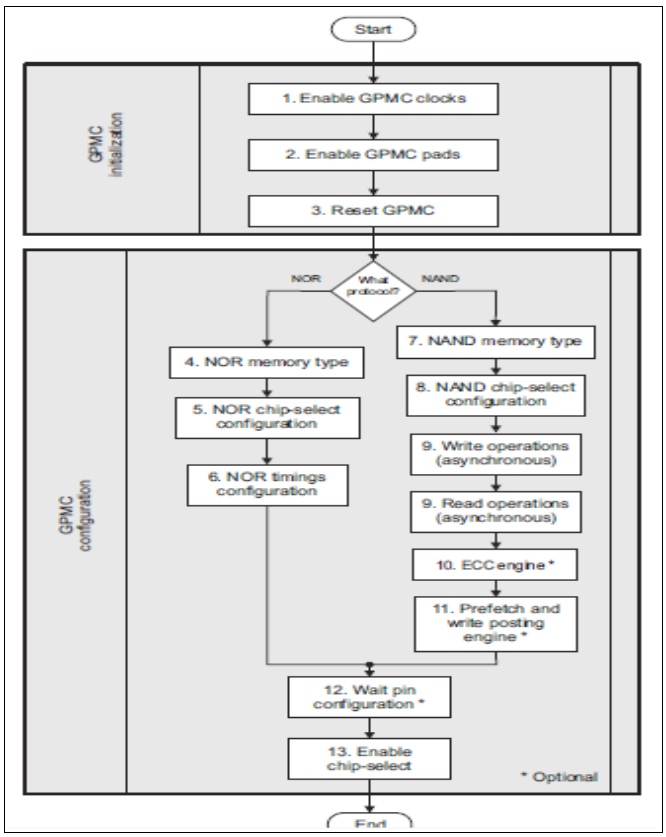


Fig. 36: *Fases de programación para implementar driver flash GPMC*

a) Sistema operativo: proceso para la programación de un driver.

En Linux un driver es tratado como un fichero, con algunas operaciones en común como read, write, open, etc.. Pero igualmente usadas como llamadas al sistema. El código que se ha desarrollado pertirá al driver dar respuesta a las siguientes 4 necesidades:

1. Inicialización: configuración inicial del periférico después del arranque
2. Exit: configuración del driver cuando el periférico haya sido suprimido
3. Operaciones de fichero normales: mencionadas en el parágrafo anterior
4. Otras config: desde la aplicación se cambia el valor de una variable del módulo

La información -desde y hacia el driver- se traspasa y se entrega en dos variables *struct* de campos con las operaciones que dispone el dispositivo y otra con campos sobre la identidad del dispositivo que a su vez contiene la estructura anterior. La aplicación de usuario que llame a la función con el nombre de una de las operaciones definidas en las estructura de operaciones activará la ejecución de la rutina indicada por el campo de su nombre. Esta será la primera parte del código del driver de la interfaz a desarrollar:

```
34  static struct file_operations sd_fops =
35  {
36      write:          sd_write,
37      read:           sd_read,
38      open:           sd_open,
39      release:        sd_release,
40      unlocked_ioctl: sd_ioctl,
41  };
42
43  static struct miscdevice sd_devs =
44  {
45      minor: MISC_DYNAMIC_MINOR,
46      name:  DRIVER_NAME,
47      fops:  &sd_fops
48  };
```

Fig. 37: *Declaración de las dos variables struct*

Después le seguirán algunas de las funciones declaradas y empleadas por las funciones de la estructura *sd_fops.* En último lugar se implementarán las funciones de operación del driver que se incluyen en la estructura y que son las llamadas de sistema. Cuando el usuario carga el módulo correspondiente al driver GPMC el código relacionado con las configuraciones de chipselect resolverán la inicialización.

También se tiene una serie de archivos cabecera en los que se definen las direcciones de los registros tanto del driver GPMC como del microcontrolador AM335x. Estos archivos se entregan como parte del material de la asignatura; pero son típicamente incluidos en las propuestas profesionales más avanzadas que pueden encontrarse en Github. No obstante, se han tenido que añadir definiciones nuevas con las direcciones físicas que el manual de Texas I. asocia con cada uno de los registros accesibles por el programador que no estaban presentes.

Las definiciones de estas cabeceras, además facilitan el trabajo al evitar que el desarrollador opere de forma constante directamente con dichos valores y en su lugar emplea denominaciones (macros) que lo hacen más intuitivo. Pero, la utilidad principal de estas definiciones es la reservarle a estas direcciones físicas un espacio de memoria (virtual) en el kernel. Pues el kernel no trabaja directamente con memoria física. Opera con espacio virtual por medio de la MMU (del uP), un traductor de memoria virtual a física. La forma de escribir o leer los registros físicos de configuración del hardware se realiza a través de este mapeo (punteros) para acceder desde el código del driver al módulo GPMC.

Es importante tener en cuenta una serie funciones y sentencias en las que el diseño se apoyará para relizar operaciones básicas como apuntar a diferentes posiciones de memoria y acceder a ellas. Las funciones y técnicas más recurridas:

- misc_register(&sd_devs): se registra el dispositivo en la categoría de miscelania de dispositivos

- ioremap(arg1, arg2): genera un puntero a la dirección física del arg1 con tamaño arg2
- hwreg(): acceso a registros
  - aux = hwreg()
  - hwreg() = aux //siendo aux → variable 32b almacenando valores hexadecimales
- printk(): permite mandar mensajes a la consola de la beaglebone desde el kernel
- udelay(): añade un retardo a la ejecución del código
- copy_to_user(), copy_from_user(): escribe dato desde origen en una posición de memoria reservada a la aplicación de usuario y viceversa. Son complementarias.

Programación GPMC NAND

	Pasos para programar el driver modo básico (requisitos mínimos obligatorios)	Pasos para programar el driver modo avanzado
	Inicialización	**Inicialización**
1	Habilitar clks	Habilitar clks
2	Habilitar pads	Habilitar pads
3	Reset GPMC	Reset GPMC
	Configuración	**Configuración**
4	Tipo de memoria	Tipo de memoria
5	Deshabilitar CS0 (sólo si se emplea CS0)	Deshabilitar CS0 (sólo si se emplea CS0)
	Configuración CS	Configuración CS
6	Write (async)	Write (async)
7	Read (async)	Read (async)
8	Habilitar CS	ECC engine
9		prefetch/write posting engine
10		Wait Pin
11		Habilitar CS

Tabla 26: *Pasos de solución avanzada Vs. solución básica*

El módulo PRCM de esta CPU basada en ARM-Cortex A8 gestiona el reset, la alimentación y el reloj de la CPU. Este bloque PRCM junto con el bloque *control module* gestionan las desconexiones (gating) y las habilitaciones de los relojes de todos los módulos del dispositivo. Uno de ellos es el reloj del módulo periférico GPMC. Para programar los campos relacionados con el control del reloj funcional del bloque GPMC debe direccionarse al grupo de registros CM_PER -perteneciente al (*clock module* del *power, reset, and clock management* (PRCM) module-, con el offset de la dirección del registro asociado al reloj del GPMC.

CM_PER	0x44E0_0000	0x44E0_3FFF	1KB	Clock Module Peripheral Registers

Fig. 38: *Mapeo del grupo de registros CM_PER (del mapa de memoria de la interfaz de periféricos L4_WKUP)*

Offset	Acronym
0h	CM_PER_L4LS_CLKSTCTRL
4h	CM_PER_L3S_CLKSTCTRL
Ch	CM_PER_L3_CLKSTCTRL
14h	CM_PER_CPGMAC0_CLKCTRL
18h	CM_PER_LCDC_CLKCTRL
1Ch	CM_PER_USB0_CLKCTRL
24h	CM_PER_TPTC0_CLKCTRL
28h	CM_PER_EMIF_CLKCTRL
2C	CM_PER_OCMCRAM_CLKCTRL
30h	CM_PER_GPMC_CLKCTRL

Fig.39 : *Offset de registros del grupo CM_PER*

Control module es otro grupo de registros destinados a controlar diferentes áreas de la CPU, entre las cuales se encuentra el multiplexado funcional de los pads/pines de la CPU.

Control Module	0x44E1_0000	0x44E1_1FFF	128KB	Control Module Registers

Fig. 40: *Mapeo y baseaddress del Control Module*

Algunos pines de la CPU están multiplexados. Poseen diferentes funciones; pero sólo pueden emplear una de ellas. Por lo tanto debe programarse el modo de funcionamiento del pin; es decir, asignarle una de las funciones para las que se dispone. Otros parámetros como el sentido (I, O, I/O) de las señales o habilitación de pull up/down internas son configurados programando estos campos: Pad Control, Mode Selection, Pull Selection y RX Active.

PRCM sostiene dos niveles de gestión de reloj dependiendo del tipo de módulo al que gestiona su reloj: protocolo esclavo y protocolo máster. El protocolo IDLE se basa en un hardware que permite al PRCM controlar el estado del módulo esclavo y le informa al esclavo mediante una petición de IDLE cuando su reloj funcional puede iniciar un *clock gating.* El esclavo puede reconocer petición de mantener el clock reposo y admitir que el módulo PRCM desactive su FCLK. El registro GPMC_SYSCONFIG permite al módulo descartar o aceptar las propuestas de *clock gating* procedentes del gestor de reloj que lo controla. Este registro forma parte del grupo de registros GPMC mapeados en en el mapa de memoria de la interfaz L3 con una dirección base de 0x50000000.

GPMC	0x5000_0000	0x50FF_FFFF	16MB	GPMC Configuration registers

Fig. 41: *Mapeo de los registros GPMC en L3*

Se accede a él (y al resto de registros de configuración del grupo del bus GPMC) apuntando a la dirección de base de los registros GPMC más el offset del registro (0x10).

De acuerdo con el "Memory Map" del manual de TI el espacio de la CUP asignada al GPMC son los primeros 512MB de este mapa de espacio del AM335x. Por lo tanto todas los accesos apuntados a las direcciones físicas inferiores a la frontera de dicho espacio (0x1FFFFFFF + 1 = 0x20000000) serán dirigidos al GPMC.

Block Name	Start_address (hex)	End_address (hex)	Size	Description
GPMC (External Memory)	0x0000_0000 (1)	0x1FFF_FFFF	512MB	8-/16-bit External Memory (Ex/R/W) (2)

Fig. 42: *L3 GPMC memory map: espacio reservado para acceso a memoria externa*

Las direcciones hasta dicha frontera se pueden construir combinando 29 líneas de bits, dado que 0x20000000=2^29. Por el momento en una primera versión de la interfaz no se pretende acceder a más de una memoria externa. Cada protocolo se implementará en versiones distintas de driver GPMC. Las memorias nand tienen la peculiaridad de no estar mapeadas y sólo se precisa tener en cuenta que el la dirección base de este espacio GPMC parte del valor 00100000h, en lugar de 0 dec, porque los primeros 000FFFFFh están reservado. Es decir, primer Mbyte se pierde y se cuenta con 511 Mbytes a repartir entre 7 CS.

En el caso de acceso a un dispositivo NOR se accede a través de un puntero que apunta a la dirección de su región definida a partir de EXTMEM_BASE (dirección física) hasta EXTMEM_BASE + 10hex (por

tanto un banco de 16 posiciones de 8 bits de memoria). Esta es una de las principales diferencias entre un driver NAND y otro NOR. La forma de transferir datos entre esas zonas y el driver se verá traducido en el código adjuntado en el anexo. En NAND este puntero no se usa.

Este puntero y otros son definidos y asignados a sus valores de espacio físico en la función *void pointers_init()*. Es a través de esta definición que se accederá a la posición de memoria que representa. En el caso de acceso a una memoria NAND no se realiza así. No se mapea. El protocolo de acceso NAND requiere un direccionamiento por fases, que se traducirá en varios accesos de escritura en el bus. Para ello el driver GPMC recibe la palabra de dirección en un registro de 32 bits del bus GPMC. Es decir, en términos del driver a implementar se accede a estos registros del mismo modo al que se acceden a los registros de configuración.

Del manual se extrae que los registros de comando y de dirección son posiciones de memoria que no almacenan los valores que se les escribe durante el acceso. Quiere decir que estos valores no se pueden capturar en forma de lectura. Pero esto no implica que el driver ignore una petición de lectura de estos registros. La implementación del driver debe garantizar que no se programen lecturas de estos dos registros para evitar resultados indefinidos. Sólo el registro de datos puede ser accedido con una lectura o escritura.

Un dispositivo NAND sólo emplea el bus de datos para compartir el traspaso de valores de comandos, dirección y datos. Lo hace de forma multiplexada, como ya se ha avanzado en apartados anteriores. La discriminación de una fase de datos, dirección o comandos se alcanza mediante la combinación apropiada de las señales de control (ALE y CLE). Como se ha dicho al inicio del apartado un acceso requiere múltiples fases de escritura previas y el manual del módulo GPMC de Texas explica qué campos de los registros de configuración deben ser programados.

Se trata de trazar equivalencias de los timings del fabricante de la memoria con los del host (lado driver) de las ilustraciones. Con el fin de adaptar los valores de los parámetros -a guardar en los campos de los registros adecuados- de los timings del fabricante, debe entenderse su función.

b) Timings GPMC

Cuando se realizan operaciones con fases de NAND COMMAND CYCLE, NAND ADDRESS CYCLE y escritura de datos:

Señal	Parámetros Escritura GPMC		
	COMMAND CYCLE	**ADDRESS CYCLE**	**INPUT DATA CYCLE**
Csn	CSONTIME, CSWROFFTIME	CSONTIME, CSWROFFTIME	CSONTIME, CSWROFFTIME
CLE	ADVONTIME, ADVWROFFTIME	ADVONTIME, ADVWROFFTIME	Desactivada
ALE	ADVONTIME, ADVWROFFTIME	ADVONTIME, ADVWROFFTIME	Desactivada
WE	WEONTIME, WEOFFTIME	WEONTIME, WEOFFTIME	WEONTIME, WEOFFTIME

Tabla 27: *Parámetros de sincronismo en los 3 tipos de ciclos de escritura*

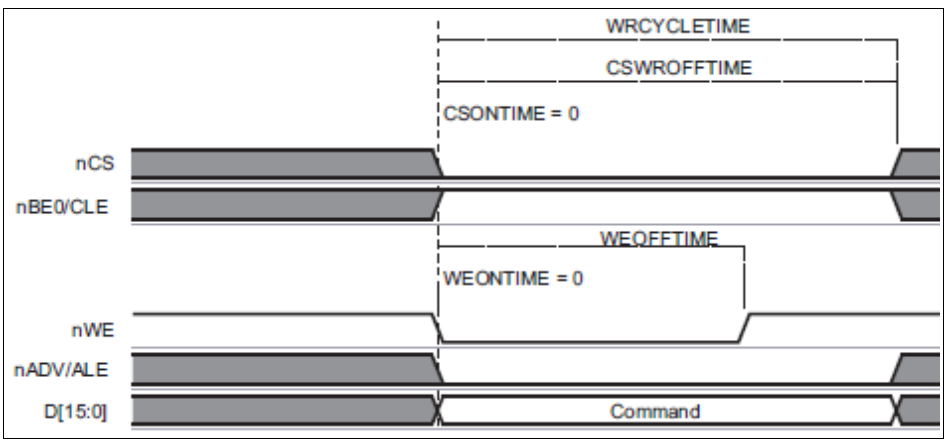


Fig. 43: *Formas de onda GPMC para el ciclo de comando*

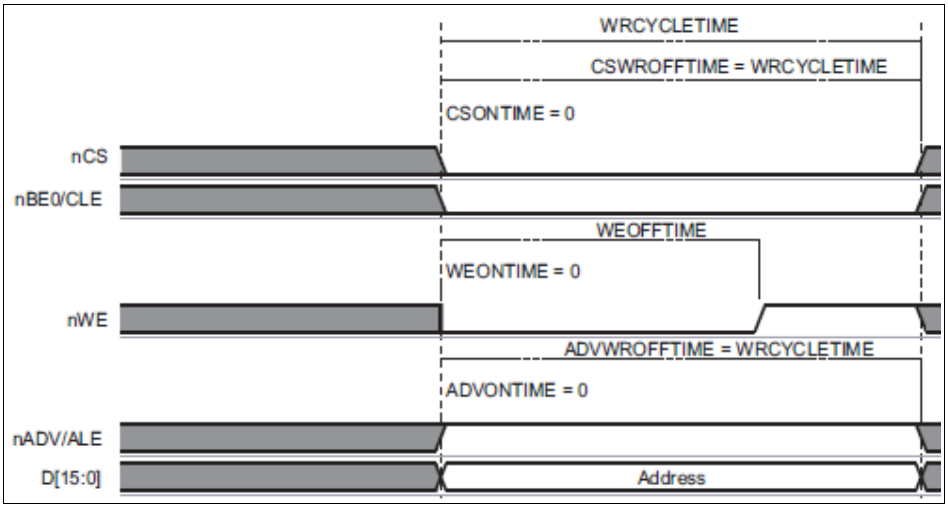


Fig. 44: *Formas de onda GPMC para un ciclo de dirección*

WEONTIME/WEOFFTIME: tiempo dentro del cual el dato es escrito en el registro (o dirección) y el bus de datos queda fijado con ALE/CLE desactivados. Puede controlar tiempo de setup y de hold antes de la activación y después de la activación de WE respectivamente.

ADVONTIME/ADVWROFFTIME: Tiempos de activación de señal y desactivación. Controla tiempo de setup y de hold de dato de dirección y de comando antes de la activación y después de la activación de la señal ADV. Esta señal se habilita una vez la dirección y comandos son fijados en el bus. Instante en que la dirección/comando son capturados ocurre entre estos dos tiempos. Así que ADVWROFFTIME debe ser superior al tiempo de activación de WE, respetando su tiempo de hold y ADVONTIME inferior al tiempo de activación de WE

CSONTIME: Típicamente usado para controlar los tiempos de setup de las señales ALE y CLE antes de que esta señal CSn se active. Es un setup hasta la activación de ALE cuando escribe. Es un tiempo respecto del tiempo de inicio del ciclo de acceso.

CSWROFFTIME: tiempo relativo al inicio de acceso que tarda en desactivarse CS. Controla el tiempo de hold de ALE y CLE después de la desactivación de CS. Tiempo de hold de Csn después de desactivar WE (flanco ascendente).

WRACCESSTIME: Se inicia cuando GPMC genera un acceso de escritura. No tiene sentido usarlo a menos que se haga como valor de ventana de tiempo en estado Wait tras una operación que requiere un tiempo de espera indicado por el fabricante de la memoria. Si se usa, la forma correcta de usarlo es de forma que el pin wait0 (RB lado GPMC) se encuentre en un estado válido dos ciclos antes de que se cumpla el tiempo WRACCESSTIME. Sino; este campo se mantiene con valor WRACCESSTIME = 0.

WRCYCLETIME: Marca el final del acceso de escritura. Definen los tiempos válidos del bus de direcciones y de datos para escribir. Cuando estos tiempos finalizan las señales de control serán desactivadas si no lo están (independientemente de los valores que les fueron asignados).

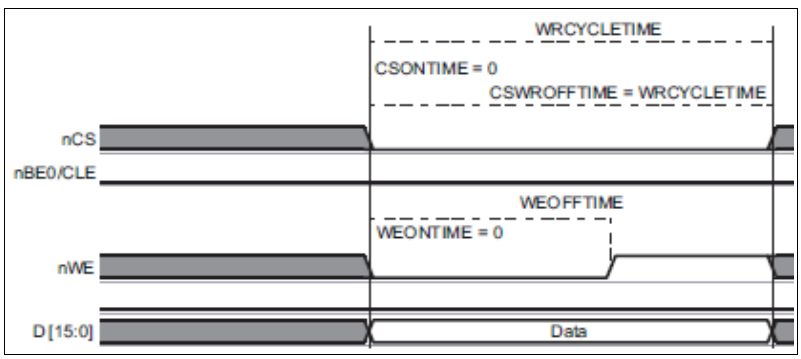


Fig. 45: *Ciclo escritura datos*

CSWROFFTIME = ADVWROFFTIME = WRCYCLETIME

Operaciones data input/output (operaciones que tienen lugar después de último ciclo de dirección excepto operaciones especiales):

Señal	Parámetros Lectura GPMC
Csn	CSONTIME, CSWROFFTIME
REn	OEONTIME, OEOFFTIME

Tabla 28: *Parámetros de sincronismo de la lectura*

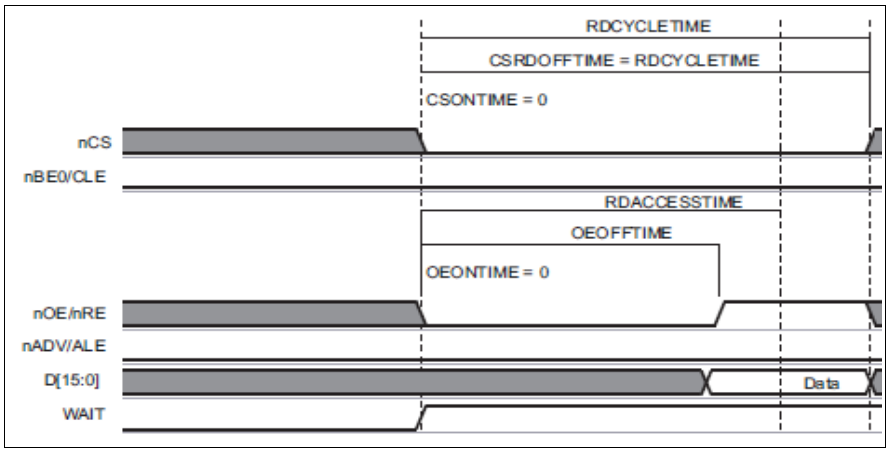


Fig. 46: *Ciclo lectura datos*

CSONTIME: Es el mismo que en escritura. Se toma como instante de inicio de read access time.

CSWROFFTIME: Desactivación de CSn debe ser programado teniendo en cuenta el valor crítico de CSn-to-data-valid ya que suele ser marcadamente superior a REn-to-data-valid (tCEA>tREA). Además hay que tener en cuenta los tiempos de hold y salida en alta impedancia tras el último ciclo de lectura.

EONTIME/EOFFTIME: Activación y desactivación de OEn. Entre ambos tiempos tiene lugar la lectura de datos con ALE/CLE desactivada. Cuando la señal se activa (flanco descendiente) la memoria deposita el dato en el bus de datos. Define el tiempo de inicio del sincronismo de lectura y el final.

RDACCESSTIME: Tiempo de latch de datos. Empieza cuando GPMC genera un acceso de lectura. Tiempo que toma a la memoria dejar el dato disponible en el bus de datos de salida. Tiempo desde el inicio de tiempo de acceso hasta el primer flanco ascendente de FCLK empleado para la captura de datos. Representa el tiempo que lleva al GPMC tomar el dato del bus. El valor debe ser programado redondeando por el valor máximo del ACCESSTIME de lectura de la memoria. Tiempo en que el dato es leído de la dirección y puesto en el bus. El dato leído es mantenido en bus y con Wait (RDY) puede controlarse el flanco efectivo del FCLK de la captura del dato en el acceso. Se recomienda que este valor se ajuste a un tiempo posterior a la desactivación de OEn. **RDACCESSTIME** puede fijarse para que el dato sea capturado efectivamente después de la desactivación de RE para tomar ventaja valor de tiempo mínimo inválido entre RE alto y dato.

RDCYCLETIME: Marca el final de los accesos. Definen los tiempos válidos del bus de direcciones y de datos para leer y escribir. Cuando estos tiempos finalizan las señales de control serán desactivadas si no lo están (independientemente de los valores que les fueron asignados). El valor mínimo debe ser CSRDOFFTIME.

CYCLE2CYCLEDELAY: tiempo inactivo que se requiere respetar entre accesos consecutivos. Las señales de control son forzadas a sus respectivos valores de restauración durante el nº de ciclos GPMC_FCLK definidos en este parámetro. Suele ser neceario incluirlos entre un ciclo de escritura y otro de escritura.

Algunas consideraciones:

tOEZ: **Al acabar el acceso** CSn y REn pasan a alta impedancia (útil el campo de busturnaround)

Write-to-read (tiempo de espera): NAND necesita un mínimo, sobretodo lectura de estado después de escritura de comando de read status. Un mínimo pulso de CSn será necesario. Usar campos → CYCLE2CYCLESAMECSEN

Csn-to-data-valid: tCEA es predominante y más lento que tREA

Ren-to-data-vaid: equivale a tREA del fabricante de la NAND.

Read-to-write: latencia de busturnaround en dos accesos seguidos → tRHW: RE to WE low = 100ns

CS0 → se habilita por defecto tras cada reset y puesta en marcha (GPMC).

Wait pin:

Especialmente en NAND resulta imprescindible controlar la disponibilidad de los datos en el bus o bien si el dispositivo ha finalizado su operación en curso. El pin RB de una memoria NAND se ha conectado al pin wait0 del bus GPMC. Este pin se ha configurado durante la inicialización para ser activo bajo, quiere decir que:

- estado ready = 1 → WAIT0 inactivo; el dato ya es válido.
- estado wait = 0 → WAIT0 activo; bus de datos inválido

Sirve para indicar la disponibilidad del dato después de una apertura de bloque/pag o que una programación del dato ha sido completada. Se tienen dos formas de ontener esta información:

1. Consultando el registro STATUS del lado host (GPMC) donde se refleja el estado del RB en el campo de *waitstatus*.
2. Consultando el registro de estado del chip de memoria mediante comando Read Status. Este también devuelve el estado de las operaciones, por lo que se emplea más para detectar errores.

Cuando las operaciones tienen tiempos de espera/ocupado cortos, el método 2 puede paralizar el acceso, por lo que se emplea el método 1. Para el resto de operaciones, como el RESET, con tiempos más altos, se emplea el método 2. Es importante no caer en el error de tratar de acceder a las memorias mientras el dispositivo se encuentra ocupado y monitoreado porque el sistema puede agotar todo su tiempo de acceso. Ante estos riesgos se propone usar el método 1, con la particularidad de deshabilitar los campos:

- GPMC_CONFIG1_i[[21] WAITWRITEMONITORING = 0
- GPMC_CONFIG1_i[[22] WAITREADMONITORING bits = 0

El pin de estado RB siempre es capturado en *wait0* y tiene su copia en el bit *waitstatus* del registro GPMC_STATUS. Pero el driver sólo puede consultarlo en un tiempo válido de la señal. Es decir, pasado el tiempo mínimo indicado por el fabricante, después del último comando escrito en la memoria (tWB).

GPMC_NAND_COMMAND_i y GPMC_NAND_ADDRESS_i son ubicaciones de 32-bits, implica que los accesos de 32 y 16 bits serán divididas en 4 ó 2 bytes si se tiene un dispositivo de 8bits conectado, de 2 ó 1 16b-word. Para comandos y direcciones de múltples fases el software del driver puede usar palabras de 16 y 32 para acceder estos registros. Pero se debe tener en cuenta la división y el esquema de ordenamiento en "little endian". Cuando una fase de comando/dirección requiere una escritura de un byte en dichos registros sólo son válidos los accesos de escritura de un byte, rellenando con el valor que indica el fabricante los MSB.

Para activar una escritura de comandos/dirección/datos en el bus de datos se realiza una escritura de datos de 16bits con el valor 0x00XX, donde X son 4bits en hexadecimal describiendo el valor del comando -a enviar al dispositivo de memoria- en uno de los 3 registros. Para activar lectura/escritura de datos pueden realizarse accesos de 32bits (con 2x16bits word de dato útil), teniendo en cuenta el esquema little-endian.

Cuando se realiza uno de estos accesos en código host ocurre la siguiente secuencia de acciones:

	Respuesta al código para escribir un ciclo comandos/dirección/datos	Respuesta al código para leer desde la memoria NAND
a	La CPU escribe el dato en el registro de comandos/dirección/datos	El GPMC realiza el ciclo de acceso físico hacia la memoria NAND.
b	La palabra es llevada desde el registro al bus en el flanco de activación de escritura WE	La memoria NAND transfiere el dato desde su registro caché hacia el bus.
c	El GPMC realiza el acceso físico hacia la memoria NAND.	La palabra leída desde el bus es capturada por el registro de datos en el flanco de activación de la señal habilitador de lectura RE.
d	El dato pasa al registro de la memoria correspondiente (caché, comando o dirección)	La CPU lee el dato de su registro de datos (suele emplear un valor típico de 100ns).

Tabla 29: *Relación de operaciones internas cuando se ejecuta acceso a un registro de datos*

c) Configuración del GPMC; valores de programación de los registros:

En este apartado se introduce un modelo de pasos para implementar la configuración de la interfaz GPMC.

Teniendo en cuenta los requisitos del lado NAND, se ha establecido el siguiente orden de acciones y que cumple el diagrama de flujo para la inicialización del driver:

Inicialización (común en los dos tipos de protocolo NAND y NOR):

1. Activación del clock funcional del módulo GPMC.
2. Activación de reset: GPMC se va a estado de reset y espera hasta que el driver salga de dicho estado (comprobando en forma de bucle).

Configuración global del GPMC (los valores dependen del protocolo y del dispositivo):

3. Deshabilitar modo IDLE para no aceptar peticiones de reposo del clock.
4. Deshabilitar contador de tiempo de espera, timeoutcontrol.
5. Deshabilitar interrupciones por máscara.
6. Seleccionar el pin wait para usar como entrada del estado de la señal RB (wait0).
7. Seleccionar la polaridad negada del pin wait0 (active low → busy).
8. Polaridad del pin de salida WP por defecto active low, invertir para desproteger el dispositivo.

Configuración del CSi (los valores dependen del protocolo y del dispositivo):

9. En caso de trabajar con el CS0 deshabilitarlo. El resto de CSi por defecto ya lo están.
10. Seleccionar el factor de escala de la latencia de las señales (x1 o x2: multiplica por dos los tiempos asignados a cada señal de control).
11. Seleccionar tipo de dispositivo: NAND flash.
12. Seleccionar tamaño del ancho del bus de datos: 16 Bit.
13. Seleccionar el protocolo de mux de datos y direcciones por el bus de datos: non-multiplexed.
14. Seleccionar tipo de acceso de operaciones: async.

Configuración de parámetros temporales (ontime, offtime) de las señales de control (depende de los datos del fabricante del dispositivo de memoria):

15. Fijar timings de CS
16. Fijar timings ADV
17. Fijar timings de WE y OE.
18. Fijar los ciclos de RD/WR (y RDACCESSTIME/WRACCESSTIME).
19. Fijar el tiempo de bus turn around (sólo entre dos accesos a diferentes CSi)
20. Fijar tiempo de *cycle to cycle* entre dos ciclos consecutivos (write-to-read).
21. Fijar el valor de base address.
22. Fijar la mask address para la medida de la región del CS.

Común en todos los tipos de acceso

23. Habilitar el CS configurado.
24. Configuración de todos los pines del bus: PINMUX config (función dedidaca a ello).
25. Desactivar WP

Implementar las funciones de lanzamiento de comandos (varía según el tipo de dispositivo):

26. Lanzar RESET
27. Monitorear señal RB (consultando registro GPMC STATUS)
28. Lanzar Read Status de la memoria
29. Lanzar Read ID
30. Borrar un bloque
31. Lanzar la programación de una página del bloque
32. Lanzar la lectura de una página

Hasta el nº 23, estos pasos se empaquetan en una única función *init_module* que será llamada cada vez que el módulo es cargado en el sistema operativo. A partir de aquí serán implementados los comandos para realizar los accesos de lectura y escritura. Se declaran e implementan las funciones encargadas de lanzar comandos como EraseBlock, readID, writePage, readPage, readStatuset desde las principales funciones de sistema. Estas funciones se basan en la adecuada secuencia de accesos con el valor de comando y direcciones correcta en los registros: GPMC_NAND_COMMAND_i y GPMC_NAND_ADDRESS_i. En el anexo referente al código puede apreciarse la solución a este target más reciente.

Se han añadido los offset de las 3 señales empleadas en NAND en la declaración de señales del archivo cabecera driver.h y se han configurado en la función pinmux.

Se define OUTPUT_EN en archivo driver.h para deshabilitar el receptor del pin, es decir disponerlo en modo sólo salida → campo Rxactive del registro pad control register declarado (soc_pin_mux_base) en archivo AM3xx.h. En tal caso de asignar ese valor al pin se debería deshabilitar pullup interno → PULLUDEN = 0.

Debe habilitarse el clock funcional del módulo si se quiere habilitar el módulo GPMC. El registro que permite configurar los campos relacionados con este reloj se encuentran en el registro CM_PER_GPMC_CLK_CTRL con la dirección de offset 30h dentro del bloque de registros CM_PER controlado por el módulo PRCM (registros del módulo de control de reloj) cuya base address es soc_prcm_regs = 0x44E1_0000 Para acceder a este registro hay que apuntar a su basseaddress más su offset y realizar una operación OR al contenido del registro para modificar únicamente el bit que interesa dejando intactos el resto de bits.

- Fijar CM_PER_GPMC_CLK_CTRL (1:0) → Campo MODULEMODE = 0x02
- Leer CM_PER_GPMC_CLK_CTRL (17:16) → Campo IDLEST = 0x03h hasta que IDLEST = 0x00

Una vez habilitado es preciso resetear el módulo como primera operación antes de iniciarlo. El campo SOFTRESET del bus GPMC_SYSCONFIG ofrece una forma de controlar por software el reset del módulo GPMC. El campo RESETDONE del registro GPMC_SYSTATUS proporciona información del estado del reset.

- Fijar GPMC_SYSCONFIG (1) = 1
- Leer GPMC_SYSTATUS(0) → Campo RESETDONE hasta que RESETDONE = 1

Una vez restaurado el sistema la primera acción es suprimir las peticiones de clock gating (IDLE) procedentes del gestor del reloj FCLK. El módulo ignorará las peticiones de IDLE, no será el modo de mayor ahorro energético; pero sí es la forma más segura de gestionar un reloj, mantenerlo siempre activo:

- Fijar GPMC_SYSCONFIG(4:3) → Campo SIDLEMODE =0x 01

Tras la restauración, el estado IDLE está activo por defecto (con un valor de 0x00). La manera más segura de operar para modificar el estado es hacerlo de forma en que sólo se modifique una vez su valor y evitar iniciar la acción con valor del campo SIDLEMODE=0x00, por ejemplo, almacenando el contenido del registro en una variable intermedia y modificándolo sobre ella. Así:

- Guardar en variable aux el valor del registro = GPMC_SYSCONFIG

- Aplicarle una and con máscara =0x00011000 para obtener el contenido actual del campo SIDLEMODE
- Aplicarle una OR con una máscara = 0x00001000 (estado no IDLE)
- Guardar la variable en registro GPMC_SYSCONFIG = variable aux

El dispositivo es muy rápido y no se va a necesitar mantener en espera al host durante ningún tiempo. Timeout control es un contador descendiente de 9 bits que puede ser habilitado y asignarle un valor entre 511 y 0.

- TIMEOUTENABLE = 0b

Tampoco se van a aceptar interrupciones. Por lo tanto se deshabilitan todas:

- GPMC_IRQENABLE = 0h

Va a haber monitoreo de la señal RB. Debe escogerse el pin que la recibirá. Después, la polaridad.

- GPMC_CONFIG1_i (17:16) → WAITPINSELECT = 0
- GPMC_CONFIG (8)→ WAIT0PINPOLARITY = 0 (por defecto) → active_low

El dispositivo NAND necesita protegerse de escrituras y borrados accidentales en sus arranques. Este modelo emplea WP. Primero:

- WRITEPROTECT= 0 (valor de restauración) → estado bajo (bloqueado)

Después de encendido y arranque:

- WRITEPROTECT= 1 → estado alto (habilitado)

Debido a que CS0 está habilitado por defecto, se recomienda deshabilitarlo antes de configurar el CS, para que tenga efecto. Se recomienda hacerlo después de seleccionar la polaridad del WP y antes de seleccionar la granularidad. Después de la configuración se habilitará como recomienda el tutorial.

Si el dispositivo es lento se pueden alargar todos los parámetros temporales de las señales de control x2. En principio se usará una granularidad de x1 ya que la respuesta de la CPU a algunas operaciones puede superar los tiempos de respuesta del dispositivo, como la lectura. Si hubiera que ajustar se modificaría posteriormente. Del paso 10 al 14 van a programarse a la vez escribiendo porque los campos de granularidad, protocolo, ancho de bus de dispositivo, tipo de acceso, modo mux del bus A/D están en el mismo registro:

- GPMC_CONFIG(13:12) → DEVISIZE = 01b
- GPMC_CONFIG(11:10) → DEVICETYPE = 10b
- El resto de campos = 0.

Los timings de CS; on/off: activación y desactivación en registro GPMC_CONFIG2_i:

- CSONTIME = 1
- CSWROFFTIME=0Fh
- CSRDOFFTIME=0Fh
- El resto de campos = 0.

Los timings de ADV (ALE y CLE); on/off: activación y desactivación en registro GPMC_CONFIG3_i:

- ADVONTIME=03h
- ADVWROFFTIME=03h
- ADVRDOFFTIME=03h
- El resto de campos = 0.

Los timings de WE/OE; on/off: activación y desactivación en registro GPMC_CONFIG4_i:

- WEONTIME=1
- WEOFFTIME=2h
- OEONTIME=1
- OEOFFTIME=2h
- El resto de campos = 0

Los timings de WRCYCLETIME y RDCYCLETIME, en registro GPMC_CONFIG5_i:

- WRCYCLETIME=0Fh
- RDCYCLETIME=0Fh
- RDACCESSTIME=02h
- El resto de campos = 0

La activación del CYCLE2CYCLEDELAY y los ciclos de busturnaround (sobretodo cuando se tienen accesos consecutivos a dos CS diferentes) se encuentran en GPMC_CNFIG6_i:

- WRACCESSTIME=0
- CYCLE2CYCLESAMECSEN=1
- CYCLE2CYCLEDELAY=6
- El resto de campos = 0

Los campos de habilitación del CS y de direccionamiento se encuentran en el GPMC_CONFIG7_i .

BASEADDRESS = codificación de base address en 6 bits (de 32 bits, equivalen a los 29:24bits).

MASK = máscara 4 bits (tamaño de la región que CS accede).

Si se emplea sólo un CSi para conectar una memoria NAND, como en esta primera versión del sistema, el baseaddress será 0x00100000 y sólo se necesitará el más pequeño tamaño, 16MB porque es una memoria no mapeada. No se apunta a una dirección de memoria externa. Así que no hace falta más para accederla. Esto hará el espacio de direcciones del CS0 ir desde 0x0010_0000 a 0x010F_FFFF. Una configuración apropiada sería GPMC_CONFIG7_0[5:0] con los bits [29:24] de la dirección completa: 0x**00**10_0000 (32 bits), por lo que 5:0 se corresponde con el valor 000000b y GPMC_CONFIG7_0[11-8] se fijará con 16MB = 1111b (la mínima).

- MASK=1111b
- BASEADDRESS = 0x00
- CSVALID=1

Para acceder al registro de configuración de cada pin del bus GPMC debe direccionarse hacia esta dirección base del control module más el offset correspondiente a su pin. Cada pin tiene asociado un registro (offsets del 804h al 89Ch) con el siguiente nombre y campos:

6	5	4	3	2	1	0
conf_<module>_<pin>_slewctrl	conf_<module>_<pin>_rxactive	conf_<module>_<pin>_putypesel	conf_<module>_<pin>_puden	conf_<module>_<pin>_mmode		
R/W-0h	R/W-1h	R/W-0h	R/W-0h	R/W-0h		

Fig. 47: *bits 6:0 del registro conf_<module>_<pin>*

- Del bit 31 al 7 son bits reservados y no se pueden modificar.

Las señales de control son tipo O esencialmente; pero programarlas como I/O parece no estar creando problemas. Wait0 es sólo I.

Se configuran los bits 6:0 del registro asociado a cada pin conforme se desea una configuración en Modo 0, habilitación de Input y pull up.

- GPMC_signalPin_Offset= mode 0 OR pull up OR fast OR receiverEnabled
    - Desde AD 0 hasta AD 15 y RB
- GPMC_signalPin_Offset = mode 0 OR pull up OR fast OR receiverDisabled
    - Señales de control

Se desactiva WP llamando a la función desactivarWP(), que no retorna ningún valor pero ejecuta una acción de cambio de polaridad en esta señal de salida.

Por seguridad se aguardan 200 us al llamar a la función uDelay(), definida en la librería de Linux: delay.h.

Se lanza un reset al llamar a la función resetNAND(), definida y creada en el driver, que escribe en el registro COMMAND una palabra hexadecimal de 16 bits con el valor del comando de reset:

- hwreg (soc_gpmc_base + GPMC_NAND_COMMAND_0_GPMC_NAND_COMMAND_i) = x00FF

Se revisa la señal RB durante 1ms y cuando RB = 1 (esté libre) el hilo sale de esta función de inicialización.

Hasta aquí todos los pasos anteriores ocurren sin que la aplicación de usuario haya sido ejecutada.

En la aplicación de usuario existen dos llamadas al sistema que activarán una secuencia de operaciones de borrado de bloque y escritura de página o lectura de ID, lectura de estado y lectura de página. Estas operaciones son funciones creadas en el driver y que son llamadas desde la propias operaciones de sistema write y read, init, etc.

Si las enumeramos por orden definición en el código son:

1. int DesactivarWP(void)
2. int ActivarWP(void)
3. int ResetNAND(void)
4. int borrarBloque(void)
5. int escribirPag(void)
6. int readIDcomand(void)
7. int readStatusCom(void)

Las memorias mapeadas por puntero, como es el caso NOR, pueden acceder a esas posiciones de memoria y escribir o leer en ellas directamente y de forma incremental. Usar una función copytouser con un buffer que incremente byte a byte su contenido es una solución viable. En cuanto a NAND, ocurre que el puntero no va a incrementar porque siempre va a escribir o leer desde el mismo registro. La función *copy_to_user()* no ha resuelto el acceso a estos registros y se ha solventado accediendo directamente a ellos, por el momento.

La formas de escribir comandos, direcciones o leer y escribir datos es:

- u16 command
- u32 data, address
- hwreg (soc_gpmc_base + GPMC_NAND_COMMAND(i)) = command
- hwreg (soc_gpmc_base + GPMC_NAND_ADDRESS(i)) = address
- hwreg (soc_gpmc_base + GPMC_NAND_DATA(i)) = data

- variableLectura=hwreg (soc_gpmc_base + GPMC_NAND_DATA_(i))

Los valores command que se han programado:

Command	Inicio	Cierre	Nº de Ciclos address	Nº Ciclo datos
RESET	0x00FF	-	0	0
READ ID	0x0090	-	1	(output) 4 bytes
READ STATUTS	0x0070	-	0	(outuput) 1 byte
ERASE BLOCK	0x0060	0x00D0	3	0
PAGE WRITE	0x0080	0x0010	5	<1024 words (input)
PAGE READ	0x0000	0x0030	5	<1024 words (outp)

Tabla 30: *Lista de comandos implementados en la versión del driver actual*

La dirección fija de bloque y de página son:

Byte	1er	2º	3º	4º	5º
Erase	-	-	0xCF	0x0F	0x00
Write/Read	0xFE	0x00	0xCF	0x0F	0x00

Tabla 31: *Valor de la dirección que se asigna desde el código*

- Bloque: 63 = B17:B6= 001111111
- Página:=15=PA5:PA0=001111
- Columna: 254 = CA7: CA0= 0000000011111110 con (CA11:CA8=Low)

La aplicación consiste en un código que permite la apertura del módulo al que trata como un fichero. Le sigue un bucle que alterna peticiones de lectura y escritura. Las cuales se traducirán en el driver en borrado y escritura de una página o lectura de la misma página.

d) Ejecución del software y comprobación de señales de control y bus de datos realizada con un analizador lógico.

Llegados a este punto, el código compila y se carga. No se producen errores y se carga sin problemas. Antes de proceder a la ejecución de la aplicación de usuario sencilla que permitirá activar las operaciones del driver, se realiza una lectura de las señales del bus GPMC y se confirma que las fases de la inicialización ocurren tal como se esperaba. Lo mismo sucede con cada una de las operaciones que la aplicación activa. Incluso, se comprueban variaciones de configuración del driver que responden tal y como es esperado. Se activan operaciones de escritura y lectura, mediante el lanzamiento de comandos -incluidos en el set de comandos del modelo seleccionado-. Se empieza por el comando más sencillo RESET y se continua siguiendo el orden de menor a mayor complejidad. En el próximo apartado se mostrarán y comentarán los resultados de las operaciones programadas en el driver.

3.4.- Integración de los elementos de la interfaz

Una vez que el controlador del host es funcional porque activa las señales correctamente y se comporta como es esperado se puede proceder a la interconexión física. La siguiente fase es la interconexión de todos los elementos de la interfaz y la ejecución del software tratando de acceder a la memoria de la PCB diseñada. Una parte de trabajo basada en desarrollo hardware suele requerir reajustes de algunos elementos.

Los conflictos de funcionamiento pueden proceder de la distorsión de señales recibidas ocasionada por la influencia de interferencias parásitas o un desajuste en los componentes de filtrado y acoplo.

Un valor demasiado grande o demasiado pequeño de un C o R puede implicar atenuaciones del componente frecuencial principal de la señal o ralentizaciones indeseadas de los tiempos de subida y bajada de señales de control conmutando. Para este tipo de efectos se disponen unos pines de test en

la cara top, que permitirán conectar el analizador u osciloscopio con el fin de comprobar las formas de onda transferidas.

Otra fuente de error posible es un mal entendimiento de las especificaciones del driver.

Sobretodo puede ser necesario un reajuste de los timings recomendados por el fabricante a base de prueba y error hasta obtener el resultado esperado. Suele ser una fuente de error muy común.

Por supuesto, puede superponerse más de una fuente de error.

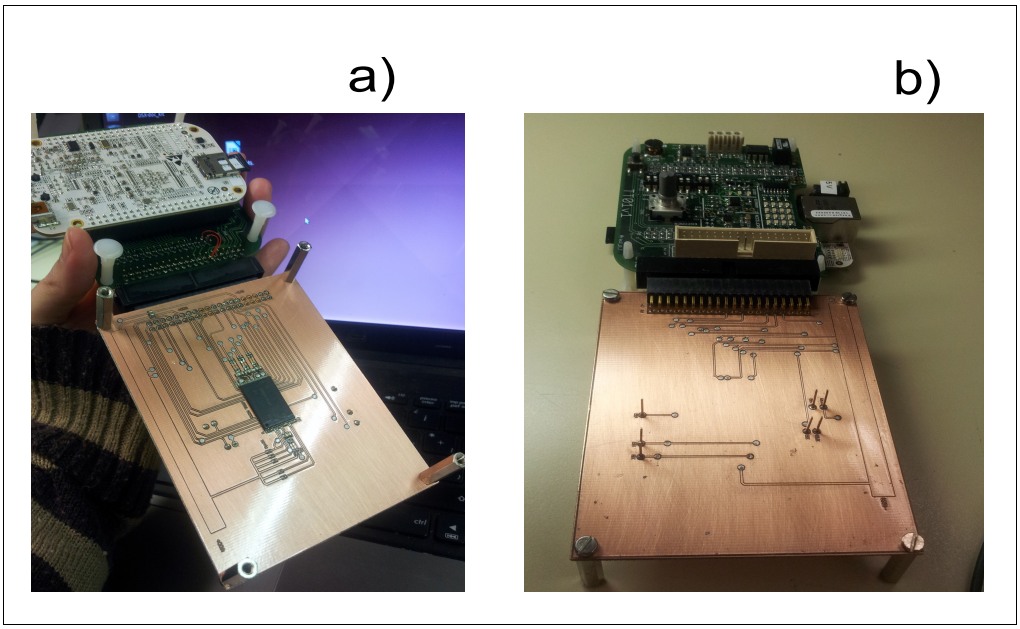


Fig. 48: *a) cara bottom (inferor), b) cara top (superior)*

4.- Resultados

En este apartado se mostrarán los resultados de las lecturas realizadas y medidas obtenidas que demostrarán el funcionamiento del driver diseñado cuando unas operaciones son demandadas por la aplicación de usuario. Una aplicación básica y sencilla llamada app demandará operaciones de lectura y escritura que activarán los envíos de comandos desde el driver.

4.1.- Comprobación del módulo GPMC.

La propuesta de valores inicial se ha ido ajustando conforme se realizaban más pruebas hasta obtener el patrón de tiempos estable y sin dilataciones indeseadas. Sobretodo los tiempos totales de acceso por ciclo, que se han visto incrementados hasta que las formas de ondas presentaban un aspecto homogéneo y equilibrado. La respuesta del driver a algunas operaciones como la lectura, -a nivel de ejecución de código-, es más lenta que la duración mínima de los ciclos de la memoria, a la vez que los tiempos de IDLE de la memoria entre operaciones se multiplican por diez en algunos casos.

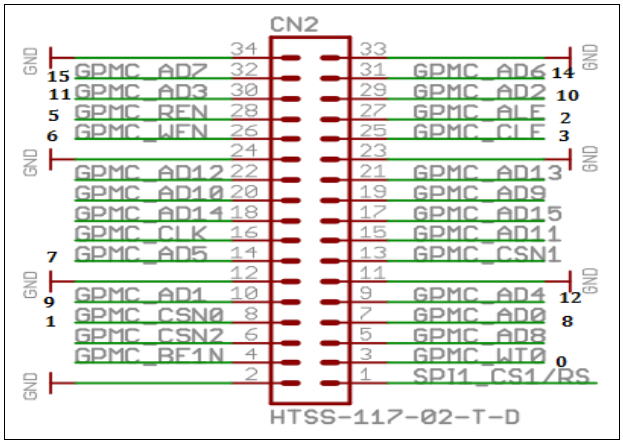


Fig. 49: *Relación CN2 con señales analizador digital*

El analizador sólo permite ver 16 canales, por lo que sólo se tiene acceso a los 8 bits menos significativos del bus IO a la vez de las señales de control. El canal 13 no está disponible y se ha cambiado por el nº 7, que estaba libre. El pin WP no se encuentra en este conector; pero se puede acceder a él por el conector CN1 usando el canal 4.

Se mostrarán los resultados de las diferentes operaciones que se han comentado en el apartado anterior. Tal como se informó el presente trabajo implementa una lista reducida de comandos, respecto de toda la gama que ofrece el fabricante para este modelo de dispositivo. Sin embargo son suficientes para permitir operaciones de borrado, escrituras y lecturas de página.

Las operaciones que implican recepción de bytes, como el caso de lectura de página, de ID o de estado, mostrarán niveles de bus desconocidos (normalmente HIGH) porque en esta sección no existe conexión con el hardware todavía.

Para estas pruebas, se han substituido los bucles de gestión de señal RB (wait0) por simples lecturas de los registros de estado en el código. Al ser una señal de entrada, las condiciones no se cumplirían nunca y se darían bucles infinitos, pudiendo colgarse el sistema durante la carga del driver. Se habilitarán los bucles condicionados durante las pruebas de acceso a la interfaz completa.

Cuando el bus GPMC no está configurado porque su driver no está cargado, la imagen de las formas de onda es la siguiente:

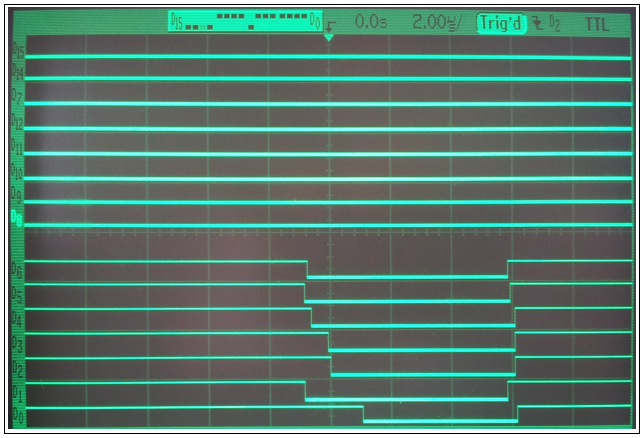


Fig. 50: *Comportamiento del driver cuando no se ha cargado en el sistema*

4.2.1.- Inicialización del driver y dispositivo periférico:

Los comandos como RESET, leer ID, leer STATUS u acciones como desactivar WP son propias de una secuencia de inicialización. Por lo tanto, pueden observarse cuando el driver se ha cargado, sin que la aplicación llegue a ejecutarse.

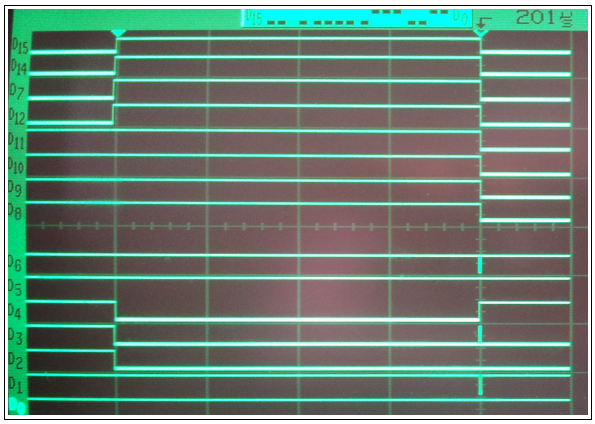


Fig. 51: *Carga del driver*

Se observa el delay, a partir del cual el controlador host permanece en espera antes de habilitar el permiso de escritura y el primer comando permitido RESET y el resto de comandos de inicialización de la memoria.

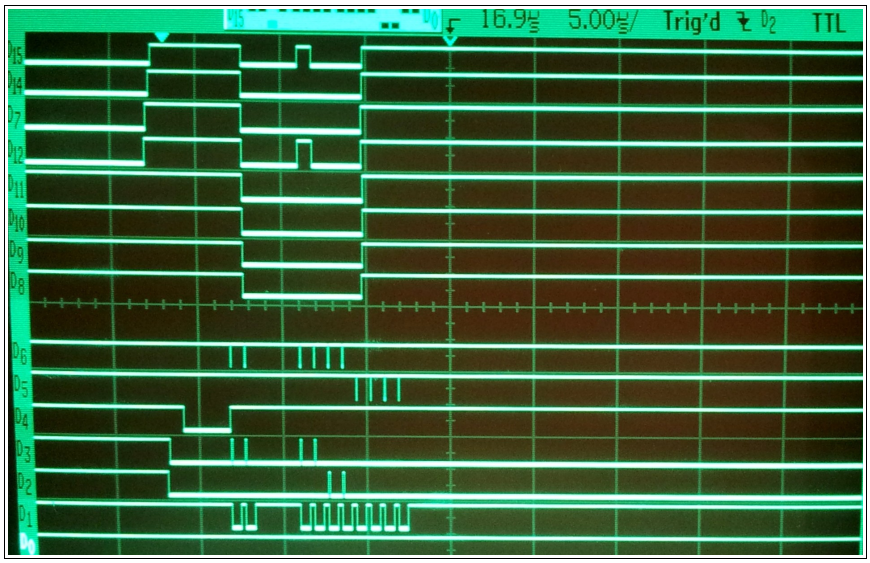


Fig. 52: *Zoom de la imagen anterior: secuencia de inicialización del driver*

Se observa como WP está activada (low) cuando se inicia el driver. Pero se desactiva insantes antes de lanzar el comando RESET (7:0). El time division es elevado y no se aprecia; pero existe un tWW entorno a los 100ns.

Otro detalle que hace falta mencionar es que todas los accesos que el driver hace con cada ciclo de acceso son 2x16 bits. Los registros que emplea son espacios de 32b y captura y escribe su contenido completo dividiéndolo en dos accesos de 16b. Por eso, el comando de RESET se presenta con un ciclo de FFh en el bus y otro 00h en segundo lugar. La línea 3 copia la señal de CLE de ciclo de comando correctamente.

Después aparece un delay de espera hasta que termine el tiempo tRST antes del siguiente envío de comando: read ID con command = 90h, address = 00h y 4 ciclos de lectura (4 palabras) en estado alto porque aún no se tiene feedback con el periférico. La línea 6 copia WE y la línea 5 copia OE con la sincronía de acceso correcta. La línea 2 marca la activación del ciclo de dirección, ALE.

4.2.2.- Operación de escritura:

La próxima sección comprobará la capacidad de realizar la escritura correcta, del módulo GPMC:

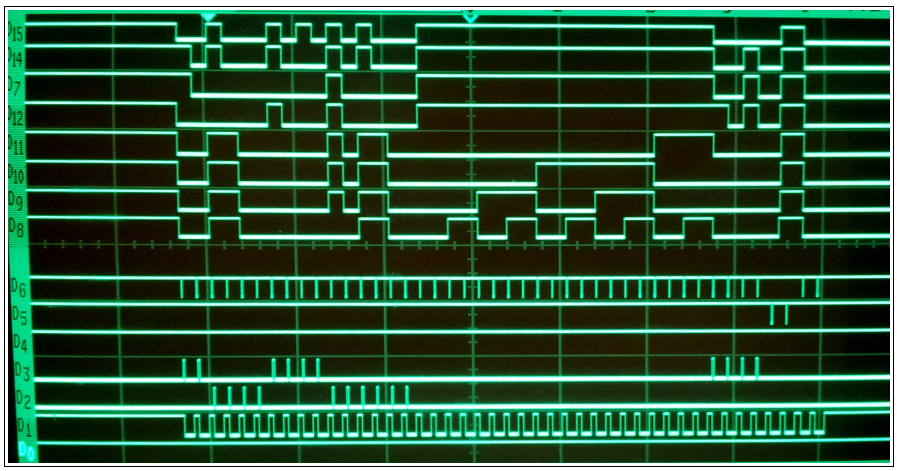


Fig. 53: *secuencia completa de una escritura de página*

Esta secuencia refleja un comando de inicio de borrado (60h), 4 bytes (3 bytes útiles y uno vacío) de dirección de bloque y otro comando de cierre (D0h). Acto seguido se aprecia un nuevo inicide comando = 80h, seguido de 5 ciclos de dirección (5 útiles una vacía) equivalentes a una dirección de columna, de página y de bloque con el siguiente orden de aparición en el bus:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	***xFE***
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	***x00***
3	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	1	***xCF***
4	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	***x0F***
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	***x00***

Tabla 32: *Direccionamiento NAND: Representación numérica de los valores del bus durante los ciclos de direccionamiento de la imagen anterior (Horizontal Bits del bus de datos/ Vertical número de ciclo)*

- Sin color: 0 obligatorio.
- Verde: Columna =nº 254
- Rosa: Página = nº 15
- Amarillo: Bloque =nº 63

Le sigue una escritura de 10 palabras. Se ha acotado a diez para apreciar una secuencia completa en el analizador; pero en realidad una escritura de página equivale a 1024 iteraciones accediendo al registro de datos del driver. Cuando esta escritura concluye le sigue el comando de cierre (10h). Todo acceso de escritura de página concluye con una revisión del estado del bit RB y del bit de estado de la operación (si se habilita la función ECC), por ello le sigue un comando (70h) de petición de estado y 1 lectura para capturar el resultado.

En último lugar se aprecia una escritura en registro de datos con el contenido del registro de estado a modo de depuración, para comprobar en futuras pruebas el valor del registro, por ende el estado de la señal RB. Es una operación sin importancia que se cambiará por una iteración condicionada a los cambios en dicha señal, mientras se aguarda a que la operación interna de la NAND concluya antes de seguir con nuevas operaciones.

Las líneas 15:8 representan los bits menos significativos del bus I/O.

4.2.3.- Operación de lectura:

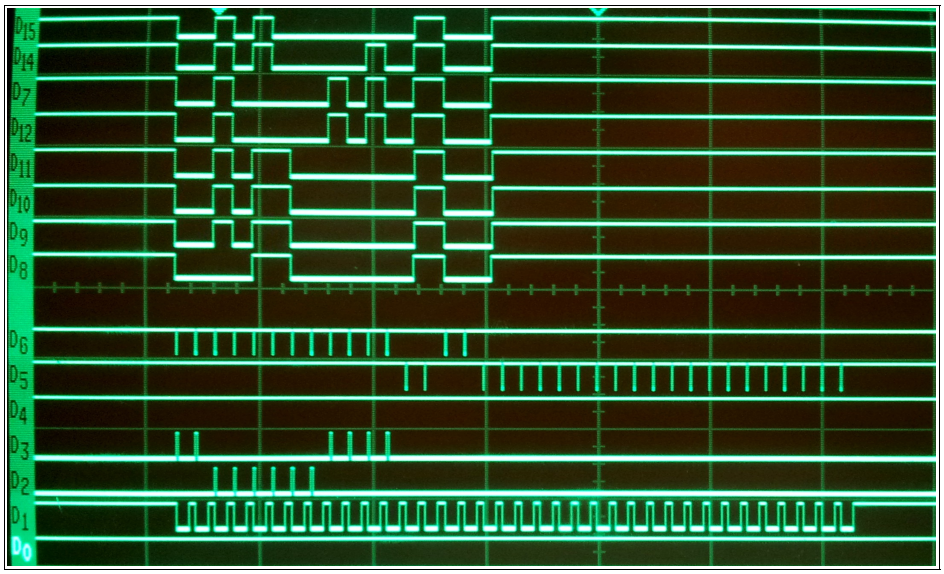


Fig. 54: *secuencia completa de una lectura de página*

Cuando la aplicación solicita una lectura el módulo activa la secuencia de la imagen. Por orden: comando de lectura (00h), 6 ciclos de dirección (5 de información útil, el sexto será ignorado), le sigue el comando de cierre(30h), luego se activa el comando de petición de estado y se activa una lectura de datos con su estado (70h) antes de que se active RB (pues el dispositivo necesitará un tR para realizar la operación internamente). La siguiente acción es una serie de 18 accesos de lectura que representan la captura de los datos solicitados.

Las señales de control se están comportado tal y como se espera. ALE y CLE se activan y desactivan durante la activación de CS, como también hace WE, que se activa dentro de ALE/CLE con los márgenes de setup y hold respetados. Lo propio se puede aplicar de RE.

Se aprecia el CYCLE2CYCLEDELAY que permite respetar los tiempos entre accesos write-to-read y read-to-write.

4.2.- Integración

En esta sección se pretende testar la conexión e intercambio de datos con el periférico; es decir se comprueba el hardware diseñado. Así que para los intentos de acceso empieza con una petición de ID

al registro de la memoria. El driver se modifica y se anulan fragmentos del código. Si se obtiene el nº de ID, se prosigue con accesos cada vez más complejos y de mayor nº de ciclos.

4.2.1.- Leer ID:

Se han rectificado el código fuente de modulenand con los tiempos de delay reales y con chequeo de la señal RB. La única operación que puede activarse desde la app, después de las operaciones necesarias de inicio, es el comando read ID.

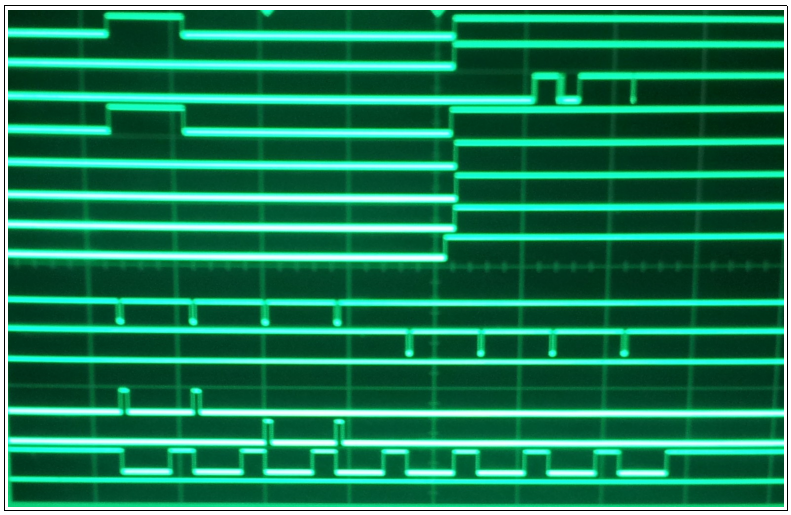


Fig. 55: *Secuencia de accesos en la interfaz para leer ID de la NAND*

Aunque parezca que la secuencia de accesos se produzca dentro de lo esperado, los datos recibidos distan de parecerse a 4 bytes de ID de Micron, ya que el primer byte bería ser = 0 0 1 0 1 1 0 0 2Ch (ID de la marca), el 2º bytes = 1 1 0 0 1 1 0 0 Cch (MT29F4G16ABADA 4Gb, x16, 3.3V), etc si sigue leyendo la lista de parámetros que ofrece micron en su datasheet. Además de que la señal RB permanece siempre libre, esto da una pista sobre que puede no estar realizando operaciones internas o bien está ignorando las peticiones. En este punto las causas son desconocidas. Se necesitan más pruebas para descartar causas y seguir valorando posibles nuevos ajustes.

5.- Conclusiones:

Del presente trabajo se ha constatado que seguir las instrucciones del fabricante es un punto de partida para deducir los valores de los campos de configuración de timings en un driver GPMC. Son útiles en una primera aproximación. A partir de ahí existe un compromiso entre las dos partes, GPMC y el dispositivo de memoria. En la práctica los tiempos del fabricante no son únicos ni exactos. Por lo general el sincronismo se perfecciona y ajusta mediante un proceso de testeo posterior. Por lo tanto, existen diferentes combinaciones posibles y válidas, mientras se respete la sincronía de señales dentro de sus máximos y mínimos. Es cierto que, aunque unas soluciones son funcionales y correctas, no significa que estas van a ser las más óptimas. Dar con ellas requiere de una dedicación práctica exaustiva y un conocimiento previo importante del hardware tanto del host como del periférico.

Se ha logrado una primera versión de interfaz software funcional que además, ofrece la posibilidad de mejorarse, ampliarse y optimizar sus timings en futuros desarrollos. No obstante estamos satisfechos con los resultados obtenidos ya que podemos reconocer en ellos los patrones de señales de un sincronismo correcto. Los comandos, ciclos de dirección y de datos son activados con los tiempos de seguridad mínimos que exige el fabricante.

La solución propuesta por este proyecto podría llevar a la apariencia estructural del host una combinación entre RAW y e·MMC, que emplea el bus GPMC en lugar de MMC para comunicarse con el host. En realidad, la actual solución pretende dejar abierta la puerta a la posibilidad de incorporar en un futuro el tratamiento de estas dos técnicas. Una vez obtenida y verificada una interfaz funcional con un hardware accesible -objetivo actual-, en un futuro; el desarrollador puede olvidarse de la problemática harwdare, centrando sus trabajo y atención en el software de control de errores (ECC), accesos DMA o atención a las interrupciones, por ejemplo.

Otro factor que se constata es la diferencia entre la programación de un protocolo NOR y un protocolo NAND de un controlador GPMC. Aunque NOR puede tener una respuesta más rápida y acceso de lectura/escritura más directo, NAND sigue siendo la solución adecuada frente al almacenamiento masivo y secuencial, debido a su alta densidad.

En contrapartida el lado de la memoria (hardware) todavía se encuentra en una fase de desarrollo en la que no presenta los resultados esperados y se sigue trabajando en ello.

La idea es continuar ajustando los parámetros del circuito de desacoplo y las resistencias de Pull up. Mejor dicho, todo aquello que pueda estar retardando los tiempos de subida y bajada de las señales de control. Dicho circuito se diseñó con la idea de trabajar con una frecuencia de corte cercana a los 40 MHz (T=20ns); ciclo de las señales de OEn y WEn cuando conmutan para habilitar la captura y escritura del dato de 16bits. Al haber modificado los tiempos de acceso es posible que los valores de los componentes del circuito estén influyendo negativamente en las señales. Aunque no es la única posible causa de error en el hardware. Lo que está claro es que la versión actual del hardware debe seguir su proceso de adaptación.

Este trabajo ha brindado la posibilidad de conocer las peculiaridades y diferencias entre la programación de código de una aplicación y la de desarrollar código Kernel para hardware de una arquitectura concreta aún empleando el mismo lenguaje. Ha permitido constatar la potencia del lenguaje C. Realizar un controlador software para el hardware de un sistema precisa un gran conocimiento de la tecnología subyacente y del espacio de memoria virtual por el cual la unidad MPU -de Linux en este caso- pretende accederle y configurar su comportamiento. Así mismo este trabajo ha permitido conocer las ventajas y posibilidades que ofrece trabajar en un entorno opensource, como este caso Linux, ya que existen diferentes comunidades proporcionando y publicando sus códigos, como:

- http://free-electrons.com/
- https://github.com/open-source
- http://www.linaro.org/

Gracias a que los requisitos de la aplicación no son exigentes, el presente desarrollo sobre el protocolo NAND da la respuesta suficiente a las necesidades planteadas: borrado, escritura y lectura de una página de la memoria NAND.

La presente solución es un controlador de funciones básicas y sería ineficaz si la memoria estuviera integrada en un sistema embebdido con una finalidad más específica y funciones tan críticas como exigentes: almacenamiento de gran volumen de datos, tiempos de respuesta mínimos, baja tasa de error (o nula), etc. Para ello habría que incluir el desarrollo de un control de error ECC y de otras tecnologías de balanceo de accesos para evitar fallos internos, maximizando así la vida útil de la memoria. Cuantas más funciones se deseen atribuir al controlador de memoria host y mayores requisitos de la aplicación, mayor grado de ampliación del driver será imprescible.

La aplicación de la versión actual pretende ser la de una plataforma de desarrollo sobre la cual elaborar diseños de driver para memoria flash y comprobar su funcionamiento, en contexto académico. Por lo tanto, la mejora y ampliación de funciones en el driver software quedan dentro de las posibilidades de este nuevo entorno de trabajo realizado. Ha quedado totalmente aclarada la metodología de trabajo. Los pasos de configuración a seguir y la relación de acciones a tomar para modificar los parámetros de interés han quedado guiados y documentados, lo cual facilita su adaptación, no sólo a este modelo concreto de memoria NAND sino a otros.

En cuanto al desarrollo de hardware se observa que en su diseño no sólo existen factores eléctricos y limitaciones dimensionales, sino que la radiación es un aspecto a analizar concienzudamente, sobretodo cuando se trata de transportar líneas de transmisión de datos. Aunque, en el trabajo actual no se opera con frecuencias superiores a los 100Mhz, lo cual reduce considerablemente estas influencias. En la práctica pueden aparecer factores que no se han tenido en cuenta en el diseño original o no se han determinado correctamente. La fases de prueba y error y reajuste es imprescindible. Los tiempos de subida y bajada pueden estar siendo afectados por la carga de entrada y salida del circuito. Una revisión de todos los elementos y consumos que influyen en estos nodos es imprescindible para determinar el margen de valores por los que estas señales pueden verse excesivamente ralentizadas. Este dispositivo adquirido permite corregir por software dentro de unos márgenes las pendientes de sus señales. Por lo que tal vez no sea necesario si quiera modificar los valores de las Rpull up.

Sobre el código del driver NAND trascienden varias ideas para optimizar y mejorar la presente versión, en etapas inmediatas, antes de pensar en añadir funcionalidades avanzadas. Actualmente, se sigue trabajando en ello también, aunque por ahora la versión no es lo suficiente madura. Se espera que próximamente se obtengan resultados dando solución a los siguientes ítems, por ejemplo:

- ser capaz de pasar los datos a escribir desde la aplicación
- ser capaz de devolver los datos leídos hacia la aplicación y mostrarlos por pantalla
- activar comandos especiales de forma independiente desde la aplicación
- hallar una forma alternativa y más flexible seleccionar la dirección
- ampliar el nº de comandos implementados y experimentar con lecturas/escrituras de bloques enteros

Otra posibilidad es:

- añadir un conector hembra a la PCB y replicar las señales de acceso NOR, de modo que pueda conectarse la FPGA a la PCB (y por lo tanto a la BB).
- Integrar los drivers NAND y NOR en un único driver de modo que se gestionen ambos protocolos en lugar de hacerlo por separado y de forma independiente.

Otra es:

- incorporar un código de detección de errores al driver NAND
- Mejorar los accesos desarrollando código para el trabajo en prefecth mode y realizar accesos DMA

- aumentar el nivel de exigencia de la aplicación, por ej: captura de imagen y tratamiento de algún sencillo algoritmo de procesado.

6.- Bibliografía

[1] R. Pérez López. "Design of a dedicated cape board for an embedded system lab course using BeagleBone". M.S. thesis, Electrical Engineering department, Technical University of Catalonia (UPC), Barcelona, Spain, 2014

[2] Varios, "BeagleBone Black", *beagleboard.org,* 2015. [Online] Disponible: http://beagleboard.org/black [Accessed: 2016].

[3] Gerald Coley, "BeagleBone System Reference Manual", *Beaglebone.org*, Rev A6.0.0 Page, Revision 0.0 May 9, 2012

[4] Varios, "Sistema embebido", *Wikipedia*, [Online] Disponible: https://es.wikipedia.org/wiki/Sistema_embebido [Accessed: 2016].

[5] Varios,"arquitectura ARM", *Wikipedia*, [Online] Disponible: https://es.wikipedia.org/wiki/Arquitectura_ARM [Accessed: 2016].

[6] Various contributors, "BeagleBone and the 3.8 kernel", *Embedded Linux Wiki*, 2014. [Online] Available: http://elinux.org/BeagleBone_and_the_3.8_Kernel [Accessed: 2016].

[7] AM335x ARM® Cortex™-A8 Microprocessors (MPUs) Technical Reference Manual, *Texas Instruments Incorporated*, 2011.

[8] Varios, "Arduino", *Arduino,* 2015. [Online] Available: https://www.arduino.cc/ [Accessed: 2016].

[9] Varios , "Raspberry Pi", *Raspberry Pi Foundation,* 2015. [Online] Disponible: https://www.raspberrypi.org/ [Accessed: 2016].

[10] Varios, "Odroid", *hardkernel.com,* 2015. [Online] Disponible: http://www.hardkernel.com/main/products/prdt_info.php [Accessed: 2016].

[11] Varios, "Intel® Edison", *intel.com*, 2015. [Online] Disponible: http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-overview.html [Accessed: 2016].

[12] Varios , "Flash memory", *Toshiba Leading Innovation,* Semiconductor Catalog Mar. 2016 [Online] Disponible: https://www.micron.com/products/nand-flash [Accessed: 2016].

[13] Varios, "V-NAND Technology STACK CELL ABOVE CELLS, BUILD TECHNOLOGY ABOVE TECHNOLOGIES", Samsung Semiconductor, 2016, [Online] Disponible: http://www.samsung.com/semiconductor/products/flash-storage/v-nand/ [Accessed: 2016].

[14] Varios, "Raw Nands", SKHynix, 2015, [Online] Disponible: https://www.skhynix.com/eng/product/nandRaw.jsp [Accessed: 2016].

[15] Varios, "NAND Flash Memory", *Cypress Embedded in Tomorrow*, 2015, [Online] Disponible: http://www.cypress.com/products/nand-flash-memory [Accessed: 2016].

[16] Varios, "Decoupling techniques", *Analog Devices,* MT-101, Rev.0, 03/09, WK.

[17] H. Torres-Ortega, "Guía de diseño de PCB con EAGLE : introducción y recomendaciones generales", Hetpro Herramientas Tecnológicas Profesionales, 2014

[18] MT29F4G16AB NAND flash memory data sheet, *Micron Technology Inc*, 2009.

[19] Varios, "An Introduction to NAND Flash and How to Design It In to Your Next Product", *Micron Technology Inc*, Technical note, 2006.

[20] Varios, "NAND Flash Controller", *Lattice Semiconductor Corp*, Reference Design RD1055, november 2009

[21] Varios, “TI E2E comunity”, *Texas instruments,* 2016. [Online] Disponible: https://e2e.ti.com/support/arm/sitara_arm/f/791 [Accessed: 2016].

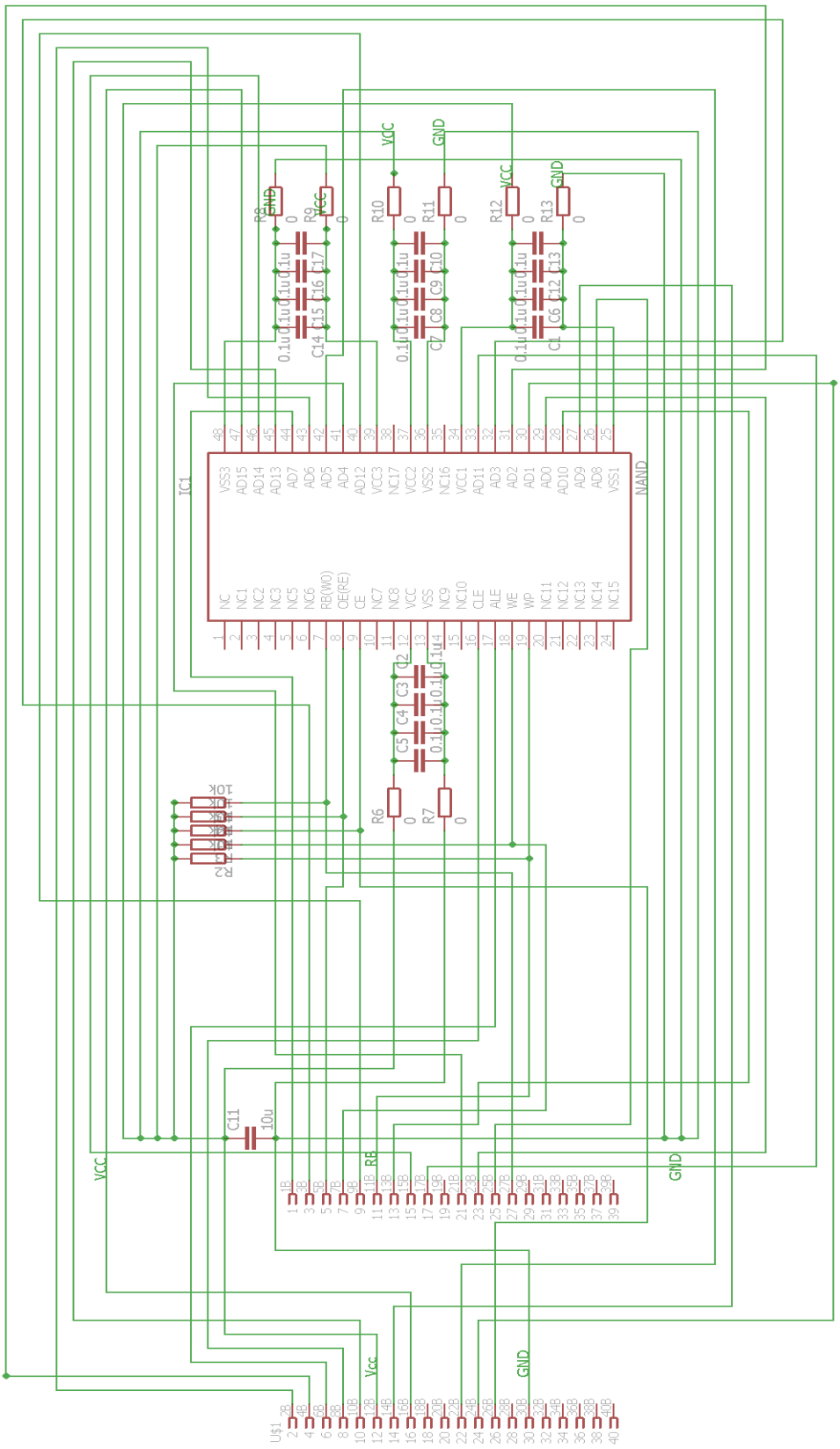
[22] Varios, “Embedded Linux development with Buildroot training”, *free electrons,* 2016. [Online] Available: http://lxr.free-electrons.com/source/include/linux/omap-gpmc.h [Accessed: 2016].

[23] Varios, “The linux kernel archives”, *The linux kernel organization*, [Online] Disponible: https://www.kernel.org/ [Accessed: 2016].

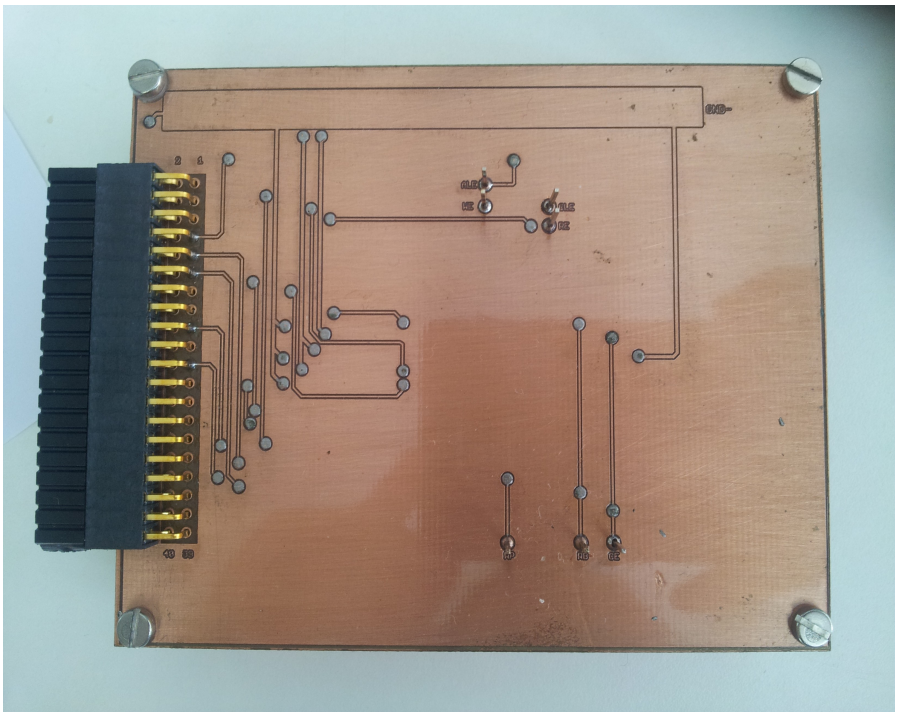
[24] Corbet, Rubini, and Kroah-Hartman, O'Reilly, “Linux device drivers; Where the Kernel Meets the Hardware”, 3rd ed, California, Usa, 2005

7.- ANEXOS

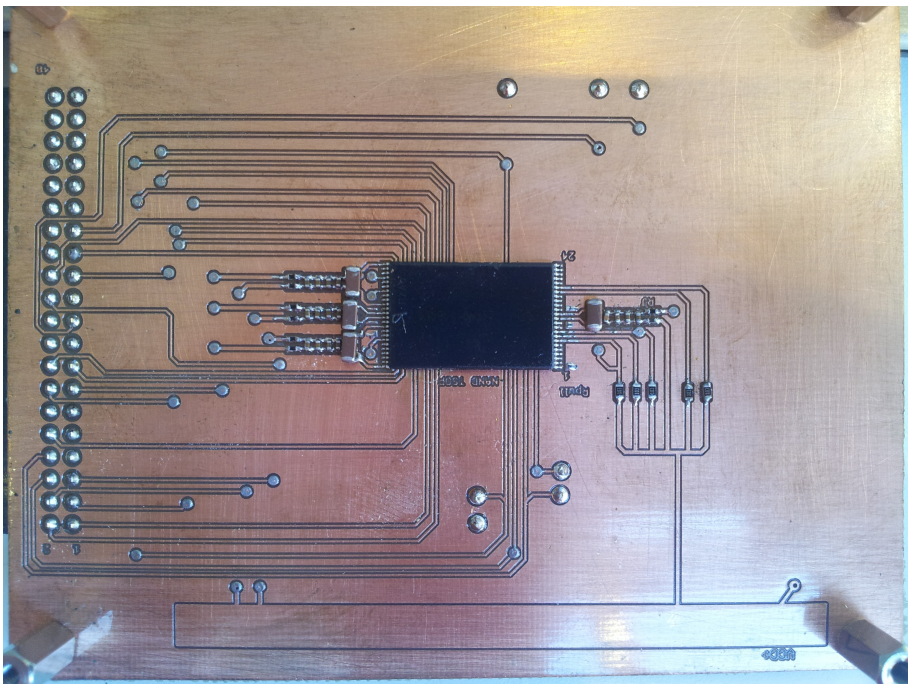
Anexo 1



ANEXO 3:



TOP



BOTTOM

ANEXO 4

```
ARCH := arm
CROSS_COMPILE := arm-linux-gnueabihf-
MAKEARCH := $(MAKE) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE)

PWD  := $(shell pwd)

default:

	$(MAKEARCH) -C $(PWD)/module M=$(PWD)
	$(MAKEARCH) -C $(PWD)/app M=$(PWD)

clean:

	$(MAKEARCH) -C $(PWD)/module clean
	$(MAKEARCH) -C $(PWD)/app clean
```

ANEXO 5

```
KSRC=/home/geni/workspace/KERNEL


ARCH := arm
CROSS_COMPILE := arm-linux-gnueabihf-

MAKEARCH := $(MAKE) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE)

EXTRA_CFLAGS := -I$(PWD)/src -I$(PWD)/include

#obj-m += module_async.o
#obj-m += module_sync.o
#obj-m += module_burst4.o
obj-m += modulenand.o

PWD := $(shell pwd)

all:
	$(MAKEARCH) -C $(KSRC) M=$(PWD) modules
clean:
	$(MAKEARCH) -C $(KSRC) M=$(PWD) clean
```

ANEXO 6

```
# Emdedded Linux Makefile
# Project: Hello
# Version 1.0 (15/11/2013)
# Paths for this project
# ./        Location for Makefile and final executable
# ./src     Location for source ".c" and object ".o" files
# ./include Location for include ".h" header files

# The following two lines are needed to generate code
# for the BeagleBone board
# Comment both lines to generate code for the PC
ARM=arm
CROSS_COMPILE=arm-linux-gnueabihf-

PWD   := $(shell pwd)

# General definitions
CC=$(CROSS_COMPILE)gcc
CFLAGS=-c -Wall -O2 $(INCLUDE)
#LDFLAGS=-lpthread

# Program specific definitions

# SOURCES must include relative paths to each C source
# from the location of the Makefile
# It is recommeded to locate all this files in a src
# directory below the directory where makefile is located
SOURCES=app.c

# EXECUTABLE define the name of the final executable
# In will be generated on the location where this
# Makefile is located
EXECUTABLE=app

# Object files to be generated
```

```
# They will be located in the same place as the source files
OBJECTS=$(SOURCES:.c=.o)

# Include file paths include general "C" include location
# and the "include" directory below the location where
# this makefile is located
INCLUDE=\
      -I. \
      -Iinclude
# Make rules
all: $(SOURCES) $(EXECUTABLE)
clean:
      rm -f $(PWD)/*.o $(PWD)/$(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
      $(CC) $(LDFLAGS) $(OBJECTS) -o $@
.c.o:
      $(CC) $(CFLAGS) $< -o $@
```

ANEXO 7

```
/*--app-- first version for GPMC NAND driver version--
Subejct: DSX
Author: Eugenia Suárez Vicente
Year: 2016 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <linux/types.h>
#include <sys/ioctl.h>
#include <string.h>           // strcmp

typedef unsigned short u16;
typedef unsigned long u32;
typedef unsigned short u8;
int main()
{       const int max = 1;//probamos con el primer elemento
        u32 MicronIDbuffer[max];
        //u16 statusWord[max];
        //u8 bufferCommand[max];
        char yesno[2];
        int i,f;
                for (i=0; i < max; i++) {
                MicronIDbuffer[i]=(unsigned long)0;
        }
        /*
                for (i=0; i < max; i++) {
                statusWord[i]=(unsigned long)0;
        }
                for(i=0; i < max; i++) {
                bufferCommand[i]=(unsigned long)0;
        }
*/
        if ((f=open("/dev/driver_port", O_RDWR))<0) {
                perror( "/dev/driver_port" );
                exit( EXIT_FAILURE );
        }
do{ //lecturas cmd --> 0, 2 y 5
//      ioctl(f,1,0);  // decirle a write() que no actualice f_pos al acabar
        //ioctl(f,0,0);

        printf("\n leer pag (y/n)?\n");
```

```
        scanf("%s", yesno);
        if (strcmp(yesno,"y") == 0) { //compara las dos cadenas
                for (i=0; i<max; i++){//tomamos solo un byte ID --> captura de 4 bytes del
reg de datos
                        if(read(f, MicronIDbuffer, 4)<0){
                                //con cada lectura obtengo un bufer de 32 bits
                                perror( "/dev/driver_port" );
                                exit( EXIT_FAILURE );
                        }
                // al final del for completo una matriz de 5 buffers
                //      printf("SDBL: micron iD = 0x%lx\n", (unsigned long)MicronIDbuffer);

                printf("leer pagina\n", (unsigned long)MicronIDbuffer);
                }
        }
        printf("\n escribir pagina (y/n)?\n");
        scanf("%s", yesno);
        //ioctl(f,1,0);//escrituras cmd --> 1, 3 y 4
        if (strcmp(yesno,"y") == 0) { //compara las dos cadenas
                //MicronIDbuffer[i]=0x09;

                if (write(f,MicronIDbuffer,4)<0) {

                                perror( "/dev/driver_port" );
                                exit( EXIT_FAILURE );
                        }
                // al final del for completo una matriz de 5 buffers
                printf("Comando borrado y read page enviados 0x%lx\n",(unsigned
long)MicronIDbuffer);
        }
}
        while (strcmp(yesno,"q") !=0);
        close(f);
        return EXIT_SUCCESS;
}
```

ANEXO 6

```
/*--modulenand-- first GPMC NAND driver version--
Subejct: DSX
Autor: Eugenia Suárez Vicente
Year: 2016 */

#include <linux/module.h>
#include <linux/fs.h>
#include <linux/miscdevice.h>
#include <linux/gpio.h>
#include <linux/slab.h>     //  kmalloc
#include <linux/uaccess.h>
#include <linux/semaphore.h>
#include <linux/delay.h> //udelay
#include <asm/io.h>         // ioremap
#include "soc_AM335x.h"
#include "am33xx.h"
#include "hw_cm_per.h"
#include "hw_gpmc.h"
#include "driver.h"

static void __iomem *soc_device_id;
static void __iomem *soc_gpmc_base;
static void __iomem *soc_prcm_base;
static void __iomem *soc_cm_device_base;
//static void __iomem *soc_extmem_base;
static void __iomem *soc_pin_mux_base;

u32 f_pos_value=0;
u32 f_pos_pos_flag = 0;
u32 f_pos_pos_value = 0;
u32 two_16_words=0;
u8  IDbyte=0;
u8  statusNAND=0;
u8 mask;
u16 command, address;
ushort csNum = 0;
//ushort csNum = 2;
static struct file_operations sd_fops =
{
    write:          sd_write,
    read:           sd_read,
    open:           sd_open,
    release:        sd_release,
```

```
    unlocked_ioctl: sd_ioctl,
};
static struct miscdevice sd_devs =
{
    minor: MISC_DYNAMIC_MINOR,
    name:  DRIVER_NAME,
    fops:  &sd_fops
};
struct semaphore mysem;
void pointers_init(void) //create pointers mapping physical address
{
    //soc_extmem_base     =   ioremap(EXTMEM_BASE, SZ_128K);
    soc_gpmc_base       =   ioremap(SOC_GPMC_0_REGS, SZ_16M);           //GPMC - base
    soc_prcm_base       =   ioremap(SOC_PRCM_REGS,0x12FF);
    soc_cm_device_base  =   ioremap(SOC_CM_DEVICE_REGS,SZ_256);
    soc_pin_mux_base    =   ioremap(AM33XX_CTRL_BASE, PAD_CTRL_SIZE);
    soc_device_id       =   ioremap(SOC_DEVICE_ID_REGS, 4);             // datasheet
}
/*****************************************************************
 * @Function: gpmc_configuration
 *****************************************************************/
uint gpmc_configuration(void)
{
    uint temp = 0;
    //enable clock to GPMC module
    hwreg(soc_prcm_base + CM_PER_GPMC_CLKCTRL ) |=
        CM_PER_GPMC_CLKCTRL_MODULEMODE_ENABLE;
    //check to see if enabled
    while((hwreg(soc_prcm_base + CM_PER_GPMC_CLKCTRL) & CM_PER_GPMC_CLKCTRL_IDLEST) !=
        (CM_PER_GPMC_CLKCTRL_IDLEST_FUNC << CM_PER_GPMC_CLKCTRL_IDLEST_SHIFT));

    //reset the GPMC module
    hwreg(soc_gpmc_base + GPMC_SYSCONFIG ) |= GPMC_SYSCONFIG_SOFTRESET;
    while((hwreg(soc_gpmc_base + GPMC_SYSSTATUS) & GPMC_SYSSTATUS_RESETDONE) ==
        GPMC_SYSSTATUS_RESETDONE_RSTONGOING);
            //todos los registros quedan con sus valores por defecto

     //Configure to no idle - manejo de energía del gpmc
    temp = hwreg(soc_gpmc_base + GPMC_SYSCONFIG);
    temp &= ~GPMC_SYSCONFIG_IDLEMODE;
    temp |= GPMC_SYSCONFIG_IDLEMODE_NOIDLE << GPMC_SYSCONFIG_IDLEMODE_SHIFT;
    //todos los cambios intermedios en temp, no en el registro
    hwreg(soc_gpmc_base + GPMC_SYSCONFIG) = temp;
    //irqs masked
```

```
    hwreg(soc_gpmc_base + GPMC_IRQENABLE) = 0x0;
    //disable timeout
    hwreg(soc_gpmc_base + GPMC_TIMEOUT_CONTROL) = 0x0;
        //select waitpin=waitpol = WP = 0
        hwreg(soc_gpmc_base + GPMC_CONFIG) |= (GPMC_CONFIG_NANDFORCEPOSTEDWRITE_FORCEPWR <<
        GPMC_CONFIG_NANDFORCEPOSTEDWRITE_SHIFT); //--> if activated post = pipeline 8 writebuffer
        //disable CS0
        //hwreg(soc_gpmc_base + GPMC_CONFIG7(csNum)) |=  (0x0 << GPMC_CONFIG7_0_CSVALID_SHIFT);

      //configure for NAND and granularity x1//NAND non mux enable 16b acceso asinc
    // CONFIG 1 ..
    hwreg(soc_gpmc_base + GPMC_CONFIG1(csNum)) = (0x0 |
        (GPMC_CONFIG1_0_DEVICESIZE_SIXTEENBITS << GPMC_CONFIG1_0_DEVICESIZE_SHIFT ) |
        (1<<GPMC_CONFIG1_0_TIMEPARAGRANULARITY_SHIFT)|
 // (GPMC_CONFIG1_0_DEVICESIZE_EIGHTBITS << GPMC_CONFIG1_0_DEVICESIZE_SHIFT ) |
//(GPMC_CONFIG1_0_ATTACHEDDEVICEPAGELENGTH_FOUR<< GPMC_CONFIG1_0_ATTACHEDDEVICEPAGELENGTH_SHIFT ) |
(GPMC_CONFIG1_0_DEVICETYPE_NANDLIKE << GPMC_CONFIG1_0_DEVICETYPE_SHIFT ));
// start timing config
    // config 2 .. chip select assert/deassert times
    hwreg(soc_gpmc_base + GPMC_CONFIG2(csNum)) = (0x0 |
        (0x01) |                                    //CS_ON_TIME
          (0xF << GPMC_CONFIG2_0_CSRDOFFTIME_SHIFT) | //CS_DEASSERT_RD trp + tcoh + 11 = (trp -trea) +
tcea + (trhz - tchz) = 81
        (0xF << GPMC_CONFIG2_0_CSWROFFTIME_SHIFT)); //CS_DEASSERT_WR

    // config 3 .. latch enable assert and de-assert
    hwreg(soc_gpmc_base + GPMC_CONFIG3(csNum)) = (0x0 |
        (1 << GPMC_CONFIG3_0_ADVONTIME_SHIFT) |     //ADV_ASSERT  **********************
        (0x3 << GPMC_CONFIG3_0_ADVRDOFFTIME_SHIFT) |  //ADV_DEASSERT_RD ***********
        (0x3 << GPMC_CONFIG3_0_ADVWROFFTIME_SHIFT));  //ADV_DEASSERT_WR******

    // config 4 .. output enable / read write enable assert and de-assert
    hwreg(soc_gpmc_base + GPMC_CONFIG4(csNum)) = (0x0 |
        (0x1 << GPMC_CONFIG4_0_OEONTIME_SHIFT) |    //OE_ASSERT
        (0x2 << GPMC_CONFIG4_0_OEOFFTIME_SHIFT) |   //OE_DEASSERT******************
        (0x1 << GPMC_CONFIG4_0_WEONTIME_SHIFT)|     //WE_ASSERT ******************
        (0x2 << GPMC_CONFIG4_0_WEOFFTIME_SHIFT));   //WE_DEASSERT**********************
    // Config 5 - read and write cycle time
    hwreg(soc_gpmc_base + GPMC_CONFIG5(csNum)) = (0x0 |
        (0xF << GPMC_CONFIG5_0_RDCYCLETIME_SHIFT)|  //CFG_5_RD_CYCLE_TIM XXX****
        (0xF << GPMC_CONFIG5_0_WRCYCLETIME_SHIFT)|  //CFG_5_WR_CYCLE_TIM XXX
        (0x2 << GPMC_CONFIG5_0_RDACCESSTIME_SHIFT));//CFG_5_RD_ACCESS_TIM XXX*****
 // Config 6 .. bus turnaround delay, etc
    hwreg(soc_gpmc_base + GPMC_CONFIG6(csNum)) = (0x0 |
        (8 << GPMC_CONFIG6_0_BUSTURNAROUND_SHIFT)|
```

```
        (GPMC_CONFIG6_0_CYCLE2CYCLESAMECSEN_C2CDELAY <<  GPMC_CONFIG6_0_CYCLE2CYCLESAMECSEN_SHIFT)|
        (6 << GPMC_CONFIG6_0_CYCLE2CYCLEDELAY_SHIFT) |  //CYC2CYC_DELAY whr - tch - 11 = 44
        (0 << GPMC_CONFIG6_0_WRDATAONADMUXBUS_SHIFT)|   //WR_DATA_ON_ADMUX XXX --> valor 4 originalmente
con mux databus
        (0 << GPMC_CONFIG6_0_WRACCESSTIME_SHIFT));      //CFG_6_WR_ACCESS_TIM


       //ACTIVAR C2C ENTRE W-R Y KIZAS TURN AROUND
       //bus turnaround??? 100ns? latencia entre accesos read to write
         //HABILITAR cs0
       hwreg(soc_gpmc_base + GPMC_CONFIG7(csNum)) =
        (0x0 << GPMC_CONFIG7_0_BASEADDRESS_SHIFT) |    //CFG_7_BASE_ADDR
        (0x1 << GPMC_CONFIG7_0_CSVALID_SHIFT) |
        (0x0f << GPMC_CONFIG7_0_MASKADDRESS_SHIFT);   //CFG_7_MASK
       hwreg(soc_cm_device_base+CM_CLKOUT_CTRL) = 0xA1; // Send L3 clock/5 = 40 Mhz to CLKOUT2
       return 0;
}


/*****************************************************************
 * @Function: pin_mux_configuration
 *****************************************************************/
int pin_mux_configuration(void)
{
    hwreg(soc_pin_mux_base + GPMC_AD0_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD1_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD2_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD3_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD4_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD5_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD6_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD7_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD8_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD9_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD10_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD11_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD12_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD13_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD14_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_AD15_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_CSN2_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_ADVn_ALE_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_BE0n_CLE_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_BE1n_OFFSET)  = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_CLK_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_OE_REn_OFFSET)    = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_WEn_OFFSET)   = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
```

```
  //  hwreg(soc_pin_mux_base + GPMC_CSN1_OFFSET) = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;//
    hwreg(soc_pin_mux_base + GPMC_CSN0_OFFSET) = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_W0_OFFSET) = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
    hwreg(soc_pin_mux_base + GPMC_WP_OFFSET) = OMAP_MUX_MODE0 | AM33XX_INPUT_EN | AM33XX_PULL_UP;
//registro en am33xx.h declarado
//tal vez mejor hacer RX --> outpun_en en las señales salidas
    return(0);
}
int DesactivarWP(void){

        hwreg(soc_gpmc_base + GPMC_CONFIG)|= (0x1<<GPMC_CONFIG_WRITEPROTECT_SHIFT) ;
        return 0;
}
int ActivarWP(void){
        hwreg(soc_gpmc_base + GPMC_CONFIG)|= (0x0<<GPMC_CONFIG_WRITEPROTECT_SHIFT) ;
        return 0;
}
int waitUntilReady(void){ //control GPMC wait0 status

       while((hwreg(soc_prcm_base + GPMC_STATUS) & GPMC_STATUS_WAIT0STATUS) !=
        (GPMC_STATUS_WAIT0STATUS_W0ACTIVEH << GPMC_STATUS_WAIT0STATUS_SHIFT));
              //if wait0 no es = 1, still looping
       return 0;
}
int ResetNAND(void){
       //comand=0x00FF;
       hwreg(soc_gpmc_base + GPMC_NAND_COMMAND(csNum)) = 0x000000FF;
       return 0;
}
int borrarBloque(void){
       hwreg(soc_gpmc_base + GPMC_NAND_COMMAND(csNum)) =0x00000060; //comando erase block pg 16, bloque
64
       hwreg(soc_gpmc_base + GPMC_NAND_ADDRESS(csNum)) =0x000F00CF; //16 page,63 block --> page address
will be ignored by LUN
       hwreg(soc_gpmc_base + GPMC_NAND_ADDRESS(csNum)) =0x00000000;//
       hwreg(soc_gpmc_base + GPMC_NAND_COMMAND(csNum)) =0x000000D0;
return (0);
}
int escribirPag(void) {
       //int i;
       hwreg(soc_gpmc_base + GPMC_NAND_COMMAND(csNum)) =0x00000080; //write page
       hwreg(soc_gpmc_base + GPMC_NAND_ADDRESS(csNum)) =0x000000FE;// 254 column
       hwreg(soc_gpmc_base + GPMC_NAND_ADDRESS(csNum)) =0x000F00CF;//16 page,63 block
       hwreg(soc_gpmc_base + GPMC_NAND_ADDRESS(csNum)) =0x00000000;//
       /*
```

```
        for (i=0; i < 1024; i++){
                hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =two_16_words;
        }*/
        //ini datos desactivar fragmento en pruebas reales
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF0FFF0;//data from gpmc to bus
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF1FFF1;
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF2FFF2;
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF3FFF3;
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF4FFF4;
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF5FFF5;
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF6FFF6;
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF7FFF7;
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF8FFF8;
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) =0xFFF9FFF9;
        //fin datos
        hwreg(soc_gpmc_base + GPMC_NAND_COMMAND(csNum)) =0x0010;
return (0);
}
int leerPag(void){
        hwreg(soc_gpmc_base + GPMC_NAND_COMMAND(csNum)) =0x00000000; //erase block
        hwreg(soc_gpmc_base + GPMC_NAND_ADDRESS(csNum)) =0x000000FE;// 254 column
        hwreg(soc_gpmc_base + GPMC_NAND_ADDRESS(csNum)) =0x000F00CF;//16 page,63 block
        hwreg(soc_gpmc_base + GPMC_NAND_ADDRESS(csNum)) =0x00000000;//
        hwreg(soc_gpmc_base + GPMC_NAND_COMMAND(csNum)) =0x00000030;
return (0);
}
int readIDcomand(void){
        int i;
        IDbyte=0;
//while((hwreg(soc_gpmc_base + GPMC_STATUS) & GPMC_STATUS_EMPTYWRITEBUFFERSTATUS) ==
// (GPMC_STATUS_EMPTYWRITEBUFFERSTATUS_NOTEMPTY << GPMC_STATUS_EMPTYWRITEBUFFERSTATUS_SHIFT)); //si el
bufer está lleno esperamos
        hwreg(soc_gpmc_base + GPMC_NAND_COMMAND(csNum)) =0x00000090; // read id command
        //añadir algun control de finalizacin de escritura en bus (revisar condicion)
//while((hwreg(soc_gpmc_base + GPMC_STATUS) & GPMC_STATUS_EMPTYWRITEBUFFERSTATUS) ==
  //  (GPMC_STATUS_EMPTYWRITEBUFFERSTATUS_NOTEMPTY << GPMC_STATUS_EMPTYWRITEBUFFERSTATUS_SHIFT));//si el
bufer está lleno esperamos
        hwreg(soc_gpmc_base + GPMC_NAND_ADDRESS(csNum)) =0x00000000; //address latch (escribo direccion
especial del comando readid)
        for (i=0; i < 2; i++)//2x2 access = 4 acces (1 NAND ID byte x access received)
        {
        IDbyte=hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum));//data from memory to bus
        }
return (0);
}
```

```
int readStatusCom(void)
{       statusNAND=0;
        //command=0x70;
        mask=0x40;
        hwreg(soc_gpmc_base + GPMC_NAND_COMMAND(csNum)) = 0x0070;
        statusNAND = hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum));
        statusNAND &= mask;
        //while ((hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum)) & mask)==0);//wait until ready habilitar
en test final
        hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum))=statusNAND; //this is not needed at final testing
        return (0);
}
/**********************************************************************
* @Function: gmpc_init
* This function will be called each time the module is uploaded.
* It must configure the GPMC registers (calling the gpmc_configuration()
* routine) and  it also has to adequately configure the pin multiplexation
* of the GPMC interface.
 **********************************************************************/
int gpmc_init(void) {
    u32 device_feature;
    uint statusGPMC;
   // uint i;
        //recoger el valor del reg de feat disponibles de nuestro AM335xx
    device_feature = __raw_readl(soc_device_id+DEVICE_FEATURE);
         // p.174 datasheet =  0x20FF0383 for the AM3359 0x20FF0383 for the AM3358

    printk(KERN_INFO "SDBL: Microprocessor feature reference AM3359 =  %lx\n"
         ,(unsigned long)device_feature);
    gpmc_configuration();
    pin_mux_configuration();
udelay(1);//200uS pruebas reales
        DesactivarWP();//enable write on nand device (WP = high) now we need wait for tWW before next we
high
        statusGPMC=hwreg(soc_prcm_base + GPMC_STATUS);//when read a register launched it takes about
100ns
        ResetNAND();
        udelay(3);//1000us first reset takes
        statusGPMC=hwreg(soc_prcm_base + GPMC_STATUS);//leer un reg GPMC emplea unos 100ns
        readIDcomand();//comentar durante test final y habilitar en read
        waitUntilReady();
        return 0;
}
/**********************************************************************
 * @Function: sd_open
```

```
 ***************************************************************************/
static int sd_open( struct inode* pInode, struct file* pFile ) {
        return 0;
}
/***************************************************************************
 * @Function: sd_release
 ***************************************************************************/
static int sd_release( struct inode* pInode, struct file* pFile ) {
        return 0;
}
/***************************************************************************
 * @Function: sd_ioctl
 ***************************************************************************/
static long sd_ioctl( struct file* pFile, unsigned int cmd, unsigned long value) {
        if (cmd==0){
        readStatusCom();
        }
        if (cmd==1){
        ResetNAND();
                }
        if (cmd==2){
        readIDcomand();
        }
        if (cmd==3){
        DesactivarWP();
        }
        if (cmd==4){
        ActivarWP();
        }
        return 0;
}
/***************************************************************************
 * @Function: sd_read
 ***************************************************************************/
static ssize_t sd_read(struct file *filp, char __user *buffer,
                size_t count, loff_t *f_pos)
{
        //soc_gpmc_base es un puntero entendido por el MMU como espacio de kernel
        int number=0;
        uint lecturaDataReg=0;
        uint statusGPMC= 0;
        //bool flancobajada;
        int i=0;
        if (down_interruptible(&mysem))
```

```
        return -ERESTARTSYS;
        udelay(1);
        //readIDcomand();// en test final habilitar
        leerPag();//lanzamiento comando lectura pag - modo salida de datos - datos en cache esperando host request
        //udelay(1);//esperar tR=25us y comprobar estado RDY,
        readStatusCom();//si no, comparar hasta que ready
        for (i=0; i < 10; i++)//version de lectura acotada para test con analizador
        {
                lecturaDataReg=hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum));//pasan los datos desde la columna de la memoria al bus
        }
        up (&mysem);
        return(number);
        /*      for (i=0; i < 1024; i++) //test final
        {
                lecturaDataReg=hwreg(soc_gpmc_base + GPMC_NAND_DATA(csNum));
        }
        up (&mysem);
        return(number);*/
}
/*************************************************************************
 * @Function: sd_write
 *************************************************************************/
static ssize_t sd_write(struct file *filp, const char __user *buffer,
                size_t count, loff_t *f_pos) {
        int number = 0;
        uint statusGPMC= 0;
        if (down_interruptible(&mysem))
                return -ERESTARTSYS;
        borrarBloque();
        udelay(3);//tWb + tBers =3000ms enprueba final
        readStatusCom();//check RB and chek failure bit
//      waitUntilReady();//wait for ready activar en prueba final
        escribirPag();
        udelay(3);//tProg = 600us en prueba final
        readStatusCom();//check RB and chek failure bit
         up (&mysem);
        return (number);
}
/*************************************************************************
 * @Function: sd_init_module
 *************************************************************************/
static int __init sd_init_module( void ) {
   pointers_init();
```

```
    gpmc_init();
    sema_init(&mysem, 1);
    if( misc_register( &sd_devs ) )
    {
        printk( KERN_WARNING DRIVER_NAME \
            ": The module cannot be registered\n" );
        return (-ENODEV);
    }
    printk( KERN_INFO "Module module.ko uploaded ******\n" );
    return(0);
}
/*****************************************************************************
 * @Function: sd_cleanup_module
 *****************************************************************************/
static void __exit sd_cleanup_module( void ) {
       misc_deregister( &sd_devs );
       printk( KERN_INFO "Module module.ko cleaned up ******\n" );
}
module_init( sd_init_module );
module_exit( sd_cleanup_module );

MODULE_LICENSE( "GPL" );
MODULE_AUTHOR("Digital Systems using Embedded Linux");
```

8.- Glosario

OS: Operative system.

CPU: Central Processing Unit

C: Lenguaje C

Cs: Capacitores

Rs: Resistores

SD: Secure Digital

IoT: Internet of Things

RISC: Reduced Instruction Set Computer

CISC: Complex Instruction Set Computer

ARM: Advanced RISC Machines

EEPROM: Electrically Erasable Programmable Read-Only Memory

ROM: read-only memory

RAM: Random Access Memory

SRAM: Static Random Access Memory

GCC: GNU Compiler Collection -

GPMC: General-Purpose Memory Controller

ECC: Error-Correcting Code

BCH: cyclic error-correcting code

DMA: Direct Memory Access

SoC: System-on-Chip

USB: Universal Serial Bus

SMD: Surface Mount Device

GNU: GNU is Not Unix

FPGA: Field Programmable Gate Array

BB: Beaglebone

PCB: Printet Circuit Board

UART: Universal Asynchronous Receiver-Transmitter

PWM: Pulse-Width Modulation

GPIO: General Purpose Input/Output

DTB: Device Tree Blob (Flat Device Tree)

SLC: Single Level Cell

MLC: Multilevel Cell

MMC: MultiMediaCard

LUN:logic units

uC: Micro Controller

uP: Micro Processor

PC: Personal Computer

IP: Internet Protocol

SDK: Software Development Kit